UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Centre de la Imatge i la Tecnologia Multimèdia

Creació d'un prototip de Videojoc: Fantasy Royale

Cognoms: Oña Rufí **Nom:** Jordi

Pla: 2014

Director: Ripoll Tarré, Marc

ÍNDEX

RESUM

Aquest document tracta sobre l'explicació i descripció del projecte "***Fantasy Royale***", un prototip de videojoc Multijugador Online que combina els gèneres *Battle Royale* i Estratègia per Torns amb Cartes Col·leccionables.

Els jugadors entraran dins el camp de batalla, on hauran de buscar les cartes que els hi proporcionaran les millors habilitats per a poder eliminar a la resta de contrincants i sobreviure a les adversitats del mapa. Com en tots els *Battle Royale*, la victòria se l'endurà l'últim jugador que segueixi en vida.

És un projecte realitzat conjuntament entre en **Sergio Sáez Calero**, encarregat del *Game Design*, i jo, encarregat de la Programació.

A través d'aquest document podem observar el procés de desenvolupament de cadascun dels sistemes que he implementat (Controlador de Personatge, Controlador de Mapa, Interfície d'Usuari, Sistema de Cartes Col·leccionabes, *Networking* i *Game Loop*).

És un projecte realitzat amb el motor 3D Unity usant el llenguatge de programació C#.

PARAULES CLAU

Battle Royale, Cartes, Estratègia, RPG, Multijugador, Unity, Networking

ENLLAÇOS

Al ser un projecte on hem usat assets de pagament, el repositori de Github que hem creat és privat. Així doncs, hem creat un repositori públic amb la **release** final del prototip, i aquí està l'enllaç:

https://github.com/Jordior97/Fantasy-Royale-Release/releases/tag/v1.0

També hem realitzat un **tràiler** que es pot visualitzar aquí:

https://www.youtube.com/watch?v=xl0rOkS5TAY

En aquest següent enllaç podem veure el ***gameplay*** d'una partida amb quatre jugadors:

https://www.youtube.com/watch?v=5ewxODuyq5U

ÍNDEX DE TAULES

ÍNDEX DE FIGURES

GLOSSARI

Alpha: primera versió completa de desenvolupament d'un videojoc, que conté tots els elements i característiques necessàries per avançar a la següent fase (Beta).

API (*Application Programming Interface*): conjunt de funcions que ofereix una biblioteca per a ser usat per un altre software. És una capa d'abstracció.

Asset: un recurs, un element, ja sigui visual (model, textura...), de codi, etc. usat en el desenvolupament del joc i en forma part.

Array: agrupació d'objectes.

Bug-Fixing: procés de correcció d'errors (*bugs*) visuals, de codi, etc.

Core Gameplay: la idea bàsica del joc, les mecàniques principals i accions que estarà efectuant el jugador al llarg de la partida.

HUD (Heads-Up Display): interfície del joc que mostra la informació de la partida per pantalla.

In-App Purchase: compres que els usuaris poden efectuar dins el joc o aplicació.

Latència: retard total produït dins de la xarxa, a causa de la demora en l'enviament i rebuda de paquets d'informació.

Lobby: sala virtual on es reuneixen els jugadors i es realitzen les configuracions prèvies, abans de començar la partida.

Milestone: objectiu, fita.

Mod: prové de "modification", és una extensió/modificació d'un joc existent que aporta noves característiques, funcionalitats, personatges… Són creats per *modders* (persones que s'encarreguen de realitzar aquestes extensions dels jocs).

Pathfinding: algoritmes de cerca de camins donats un punt inicial i un final.

Playtesting: prova en que usuaris juguen al joc sota unes condicions determinades i es prenen anotacions amb l'objectiu de millorar la usabilitat, jugabilitat i altres aspectes del joc en funció dels comportaments observats.

PEGI (Pan European Game Information): sistema de regulació europeu que informa sobre l'edat mínima adequada per consumir un producte.

Rendering: sistema/procés de generació i visualització d'imatges (2D o 3D).

Serialitzar: procés de guardar informació en un medi d'emmagatzematge (fitxer, etc.) i transmetre'l a un receptor.

Socket: "paquet de dades" que permet la transmissió d'informació entre dos o més ordinadors a través de la xarxa.

Scripting: sistema/llenguatge de programació que suporta scripts (programes compostos per codi que executen les tasques programades).

Streaming: retransmissió, distribució digital de contingut multimèdia.

Tag: paraula clau associada a un o més objectes per a poder filtrar-los.

1. INTRODUCCIÓ

1.1 Motivació

Som fans dels jocs de cartes col·leccionables com *Magic: the Gathering*, *Hearthstone* pel factor d'estratègia i formació de la baralla per combatre, així com també ens agrada l'adrenalina i dinamisme dels jocs *battle royale* com *Fortnit*e i *PlayerUnknown's Battelgrounds*. Així doncs, hem decidit realitzar un **prototip d'un videojoc** que mescli aquests gèneres, una idea que creiem innovadora i atractiva per als consumidors i fans d'ambós tipus de joc.

1.2 Formulació del problema

Amb el repte que ens plantegem, la creació d'un prototip de videjoc, volem suplir la "manca d'originalitat" en els jocs d'aquest tipus (*battle royale*), que bàsicament consisteixen en *shooters* on has d'aconseguir bones armes i equipament, i disparar per eliminar a tots els teus adversaris.

Així doncs, volem donar aquest toc d'innovació i originalitat mesclant el dos gèneres: joc de cartes i *battle royale*.

1.3 Objectius generals del TFG

L'objectiu global del TFG és, doncs, crear un prototip de videojoc jugable i funcional.

Aquest projecte el durem a terme conjuntament el meu company Sergio Sáez i jo.

- Jo m'encarregaré principalment de la totalitat de la **Programació** del joc.
- El meu company Sergio Sáez s'encarregarà del **Game Design**.
- L'**Art**, com no és àmbit específic de cap dels dos, hem decidit adquirir diversos *assets* de la *Unity Store* que ens permetran suplir aquest apartat.

La idea d'aquest projecte també és aprendre coneixements sobre la creació d'aquest

1.4 Objectius específics del TFG

En quant a la part de programació, podem subdividir-la en diverses àrees, que seran doncs, els objectius específics a complir:

- **Controlador de personatge:** tota aquella funcionalitat que permetrà al jugador escollir el seu personatge i controlar-lo dins el mapa: tant el moviment com totes les possibles accions que pugui fer durant i fora del seu torn (jugar cartes, adquirir-ne de noves, descartar-les, etc.).
- **Mapa:** tot el sistema que controla les caselles que formen el mapa, per a poder aplicar *pathfinding*, àrees d'efectes i rangs d'habilitats, esdeveniments aleatoris específics (aparició de cofres especials, tancament de la zona de batalla...)
- **Interfície d'Usuari:** sistema que permetrà al jugador visualitzar tota la informació de la partida i interactuar amb el joc (visualització de les cartes, vida actual, temps del torn, rang de les habilitats...).
- **Connectivitat a Internet:** funcionalitat que permetrà als jugadors connectar-se en línia i jugar dins de la mateixa partida.
- **Sistema de Cartes "Col·leccionables":** implementar tota la funcionalitat de les cartes (habilitats, àrees d'efecte, cost, etc.) i de la baralla (com adquirir noves cartes, descartar-les, etc.)
- **Game Loop:** funcionalitat del joc des de que inicia el joc fins que acaba la partida, on tornaria a poder començar el cicle (bàsicament es tracta d'un Game Manager que controli el flux del joc).

1.5 Abast del projecte

Com he comentat, el nostre objectiu és elaborar un **prototip** d'un videojoc, jugable i funcional. No ha d'estar al nivell d'una versió final d'un videojoc real desenvolupat per un estudi.

La **primera limitació** que trobem seria l'apartat d'Art. Com cap dels nostres Treballs de Final de Grau se centra en l'apartat de l'Art hem de trobar alguna via alternativa òptima que ens permeti suplir aquest àmbit, com he comentat anteriorment.

La **següent limitació** seria el nombre de jugadors simultanis que poden jugar una partida. El servei de *Network Management* que ens ofereix *Unity* només suporta un total de sis jugadors simultanis. Seria massa ambiciós i costós (tant per falta de temps com de coneixement), intentar aplicar algun altre mètode o sistema que suporti un gran nombre de jugadors connectats en línia, ja sigui comprant un servidor o algun altre mètode.

El nostre **públic**, si fos un videojoc real amb la intenció de llençar-lo al mercat, seia el mateix al que va dirigit el joc per a *smartphones* anomenat *Brawl Stars* (PEGI +3) però principalment ens dirigirem a un públic jove-adult (d'entre 12 a 25 anys), que els hi agradin els jocs d'estratègia per torns i la modalitat de joc de *battle royale*.

2. ESTAT DE L'ART

En aquest apartat ens informarem del panorama actual, de tot allò que implica la creació del prototip de videojoc que volem desenvolupar, tant a nivell conceptual (què és un Battle Royale, fonaments dels jocs de cartes, popularitat, origen...) com a nivell tècnic (eines de creació de videojocs, tècniques més comuns, patrons de programació, etc.)

Què és un *Battle Royale*?

El concepte

Aquest gènere de videojocs s'ha popularitzat molt aquests darrers anys. L'objectiu principal és molt clar: la SUPERVIVÈNCIA. Els jugadors comencen la partida en igualtat de condicions, amb l'equipament mínim, i hauran d'explorar el mapa per apoderar-se d'armes i eliminar els rivals mentre eviten quedar-se fora de la zona segura (una àrea, normalment circular, que es va tancant al voltant d'un punt aleatori del mapa a mesura que progressa la partida). Caminar per fora d'aquesta zona suposa, en molts casos, la mort, ja que l'àrea no-segura va fent mal continu als jugadors que s'hi troben. Així doncs, com es pot deduir, el guanyador serà l'últim jugador en peu.

L'origen

Aquest concepte però, no és originari dels videojocs, sinó d'una novel·la japonesa anomenada *Battle Royale* (1999), on uns alumnes d'una classe d'institut són enviats forçosament a una illa com un experiment d'una dictadura japonesa. La crua missió que han de complir és matar-se entre ells fins que només un d'ells quedi viu dins un període de tres dies. Els únics recursos amb els que compten són un mapa, racions i una arma aleatòria. Si desobeeixen o intenten escapar, moren a causa del collar explosiu que els hi han implantat.

Figura 1 - Escena de Battle Royale (1999)

Aquesta és la idea més semblant al gènere de videojocs que trobem avui en dia, ara bé, la mínima essència d'aquest concepte el podem trobar d'abans, a finals dels anys 80 en un sector diferent: la lluita lliure. I és que la WWE va inventar un esdeveniment especial de lluita lliure anomenat *Royal Rumble*, on lluitadors anaven pujant al ring cada cert temps per lluitar i intentar quedar l'últim en peu.

Figura 2 - Combat Royal Rumble de la WWE

Els primers *Battle Royale*

Dins el sector del Videojoc, els precursors en aquest gènere no van ser jocs complets, sinó *mods* de jocs existents:

- *Minecraft Hunger Games* (2012): inspirat en la sèrie de novel·les d'aquest mateix nom, és un mode de joc on 24 jugadors es troben reunits en una àrea tancada, partint del centre del mapa, on hi ha un conjunt de cofres amb recursos.
- *"PlayerUnknown" Battle Royale* (2013): modificació d'un *mod* existent (*DayZ*) del joc *Arma II*, inspirat en la pel·lícula japonesa nombrada anteriorment. La diferència principal amb la proposta anterior és la dispersió aleatòria dels recursos pel mapa, així com dels jugadors en el punt de partida.

Figura 3 - PlayerUnknown Battle Royale

Figura 4 - Minecraft Hunger Games

El BOOM del gènere

El creador del *mod* anterior va passar a desenvolupar-lo com a títol independent, i va ser llençat en accés anticipat sota el nom de PUBG (*PlayerUnknown's Battlegrounds*) el Març del 2017. Les xifres que va aconseguir (**20 milions de còpies venudes** a finals d'any, 1,3 milions de jugadors actius simultanis) van fer que s'establís com a joc definitori del gènere i han provocat que molts títols actuals implementin el seu mode *battle royale* (fent que sigui un dels gèneres més populars i vistos en el panorama actual).

Trobem una gran quantitat d'exemples: *Black Ops 4: Blackout, Paladins: Battle Royale, Fortnite, GTA Online: Battle Royale.*

Aquest èxit del PUBG, però, ha sigut superat per la companyia *Epic Games* amb el llançament d'un mode *battle royale* gratuït del seu títol *Fortnite*. Van saber aprofitar l'oportunitat i a dia d'avui compta amb **uns 40 milions de jugadors regulars**.

De l'acció frenètica a l'estratègia dels JCC

El concepte

Els JCC (Jocs de Cartes Col·leccionables) són un tipus de jocs de cartes en què aquestes estan compostes per una sèrie de característiques (nom, imatge, descripció, tipus, etc.) que les defineixen i les fan úniques. Els jugadors, a partir d'un conjunt d'elles, configuren la seva baralla per a competir (sota les regles específiques del joc) contra altres persones, que també han configurat prèviament la seva baralla personalitzada. Aquestes cartes, per a ser jugades, compten amb un cost que implica el consum d'algun tipus de recurs que normalment es recarrega a cada inici de torn.

L'origen

Ens trobem a l'any 1993, neix *Magic: the Gathering* de la mà de Richard Garfield. Aquest joc va més enllà d'un joc de cartes tradicional, inspirat en els duels entre mags de jocs de rol, on es llancen conjurs, s'invoquen criatures… Per aconseguir aquest context, no pot ser desenvolupat sobre una baralla de cartes tradicional, on cada carta sempre és fixa i té una funció, així que crearà el seu propi món i cartes úniques.

Figura 5 - Partida de Magic: the Gathering

Característiques Principals

Podem destacar dos grans pilars sobre els quals es fonamenten aquests jocs, en relació a les cartes, que els diferencien de les baralles tradicionals (com poden ser la baralla anglesa o l'espanyola):

1. Individualitat**:** cada carta creada compta amb la seva funció específica, que vindrà determinada per la descripció i text imprès en ella. Normalment compten amb un nom, un cost, un efecte, vida i atac (si són criatures), raresa i tipus.
2. Col·leccionable: aquestes cartes s'adquireixen principalment en sobres, on no se sap què tocarà. Això implica que els jugadors intercanviïn cartes en funció de les seves necessitats.

Figura 6 - Exemple de carta col·leccionable

De les cartes físiques a les digitals

Els primers JCC portats al món digital els trobem a finals dels anys 90, i són adaptacions dels jocs existents per a ordinador. Trobem *Pokémon Trading Card Game* (1996), *Magic: the Gathering* (1997), *Yu-Gi-Oh! Duel Monsters* (1998).

La principal i més important diferenciació entre els jocs físics i digitals és la possibilitat de jugar en xarxa a través d'Internet. Al ser digitals, poden aportar una experiència al jugador més audiovisual: amb efectes, animacions i partícules... que enriqueixin la seva jugabilitat.

Impacte actual

Com ha passat amb el gènere *battle royale*, una gran part dels jocs que surten al mercat han implementat les mecàniques bàsiques d'aquest gènere, com la **col·lecció i el sistema de raresa** (com més rar és un element, millors característiques i més difícil d'aconseguir) per mantenir els jugadors actius, esperant adquirir aquell element desitjat, ja sigui intercanviant-lo si és possible, o amb sort i paciència.

Amb l'aparició dels ***smartphones***, aquest gènere s'ha popularitzat en gran mesura en aquesta plataforma pel fet de poder jugar en qualsevol lloc. També són objecte de consum per a *Streaming*, essent *Hearthstone* i *Magic Arena* dels jocs populars en la plataforma *Twitch* (amb uns 30.000 seguidors diaris entre els dos títols).

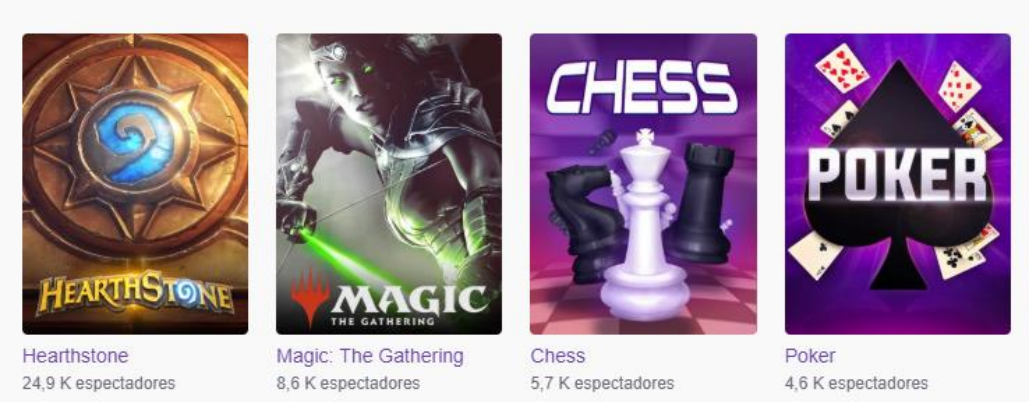

Figura 7 - Jocs de cartes en la plataforma Twitch

Creació de videojocs

Un cop som conscients del panorama actual i coneixem els dos pilars que definiran el prototip de joc que desenvoluparem, passem als detalls més tècnics.

Motors de Videojocs

Actualment, la gran majoria de jocs que són llençats al mercat són desenvolupats mitjançant un motor, que podem definir com el programa o l'eina principal que compta amb els sistemes necessaris (físiques, *scripting*, partícules, *rendering*...) que els desenvolupadors utilitzen per a crear el videojoc.

Dos dels *battle royale* més coneguts, *Fortnite* i *PUBG*, han estat desenvolupats amb *Unreal Engine*, per altra banda trobem que *Hearthstone* va ser desenvolupat en els seus inicis amb *Unity Engine*. Així doncs, analitzem breument els motors de videojocs més populars actualment:

- **Unreal Engine:** desenvolupat per la companyia *Epic Games*, és un *software* encarat a desenvolupadors d'un nivell avançat, usa el llenguatge de programació C++ i destaca per la seva superior qualitat gràfica i possibilitat de ser modificat a baix nivell. Orientat sobretot a *Shooters*, jocs realistes, de lluita i d'acció.

  Per altra banda, treballar amb ell es bastant complex, fent així que la corba d'aprenentatge sigui elevada. També compta amb menys informació i suport, i una comunitat reduïda a l'hora de cercar solucions a problemes que puguin ocórrer al llarg del desenvolupament.

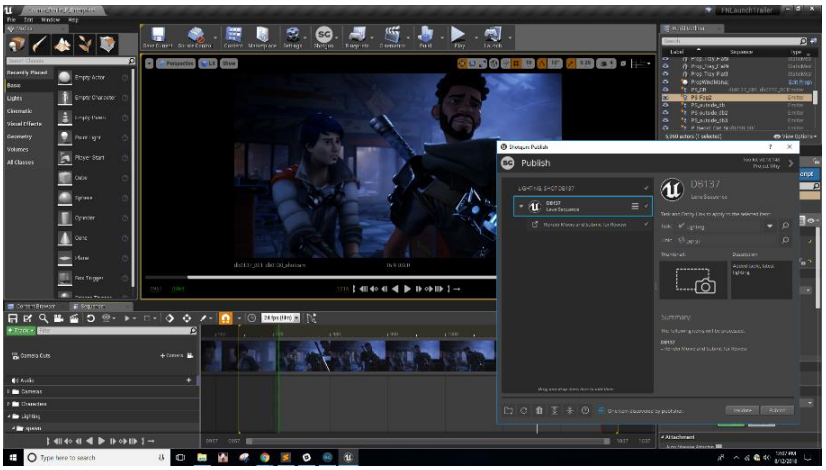

Figura 8 - Entorn de treball d'Unreal Engine

- **Unity Engine:** desenvolupat per *Unity Technologies*, és un *software* més assequible iencarat a un públic més ampli (tant per a desenvolupadors experts com aficionats). Treballa amb el llenguatge C# i destaca pel seu suport multiplataforma, una àmplia tenda *d'Assets* (gratuïts i de pagament), una gran comunitat i fonts on trobar informació, així com la facilitat d'ús i accessibilitat (una corba d'aprenentatge suau). Orientat a jocs de plataformes, *mobile games*, *casual games* i jocs d'acció.

  Per contra, aquest motor exigeix un alt rendiment a l'ordinador, i la qualitat gràfica no arriba (de moment, encara que hi ha excepcions) al nivell que pot arribar *Unreal Engine*.

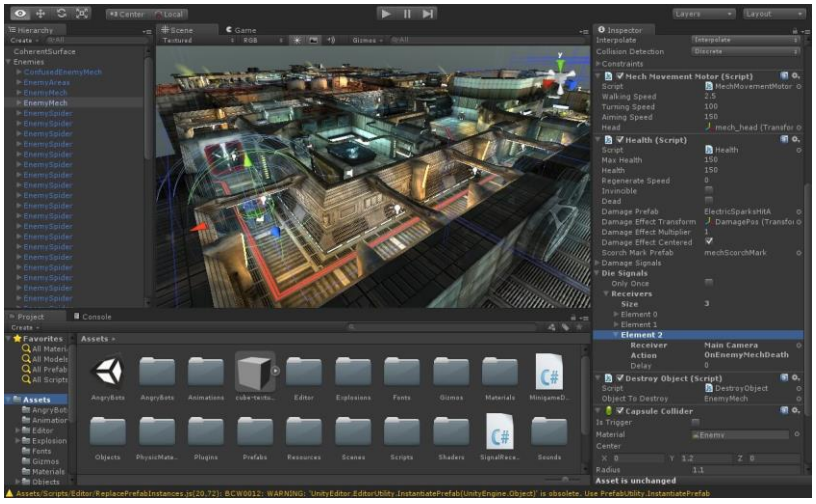

Figura 9 - Entorn de treball d'Unity Engine

Aquests dos motors són dels més populars i usats en la inústria, i en trobem d'altres com: *Godot Engine, CryEngine, AppGameKit, Cocos2D-x*, i molts més.

També cal destacar que les grans empreses com *EA, Blizzard, Rockstar...* han desenvolupat els seus propis motors de videojocs (*RAGE, Frostbite*...) per a treballar amb ells internament.

Jugar en línia, com?

La màgia dels jocs en línia es fer creure que els jugadors estan compartint un món, una realitat. Per a aconseguir-ho, s'han desenvolupat diferents tècniques:

- ***Peer-to-Peer Lockstep*:** cada màquina intercanvia informació directament l'una amb l'altra, ja que estan connectades en una topologia de malla. La idea bàsica és que les accions dels jugadors són enviades a totes les màquines connectades, que les reproduiran amb exactitud dins de la seva instància de joc (partint del mateix estat inicial) per a que tots els jugadors visualitzin el mateix estat del món en el que es troben. Aquesta tècnica s'usava inicialment en els jocs RTS (*Real-Time Strategy*), composats per torns i accions/comandes (atacar, moure's, curar-se, construir...).

  Ara bé, aquesta aproximació te una sèrie de limitacions:

  - El joc ha de ser completament determinista: totes les accions han de tenir unes conseqüències clares i específiques, ja que en cada una de les màquines s'ha de reproduir exactament la mateixa reacció (per a mantenir al idea de món consistent i compartit).
  - Problemes de connexió: per assegurar aquesta sincronització entre totes les instàncies de joc, s'ha d'esperar que totes les comandes siguin enviades a totes les màquines abans d'efectuar el torn. Així doncs els jugadors experimentaran el temps de retràs del jugador amb menys connexió (que serà el que tardarà més en rebre/enviar la informació a través de la xarxa).
  - Partir d'un estat inicial: per sincronitzar totes les instàncies de joc cal que tots els jugadors comencin en el mateix estat inicial. Així doncs, és habitual en els jocs d'aquest tipus que els jugadors hagin d'unir-se prèviament a un *lobby* abans de començar a jugar.

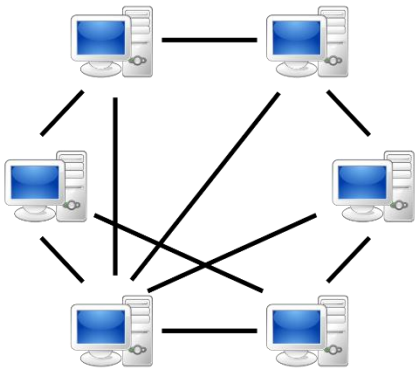

Figura 10 - Xarxa Peer-to-Peer

- **Client/Servidor:** aquest model, en comptes de que cada màquina processi una instància de joc pròpia i es comuniquin directament entre elles, fa que es comuniquin amb una sola màquina, anomenada servidor. Aquest servidor executarà la instància principal del joc, i cada client actua com a "finestra" que mostra una aproximació del joc. Idealment no s'executa codi localment, sinó que el client envia la informació de les accions realitzades pel client al servidor. Aquest el que haurà de fer serà actualitzar l'estat del joc i enviar la resposta a tots els clients. Amb aquesta tècnica tenim la possibilitat d'implementar entorns no deterministes.

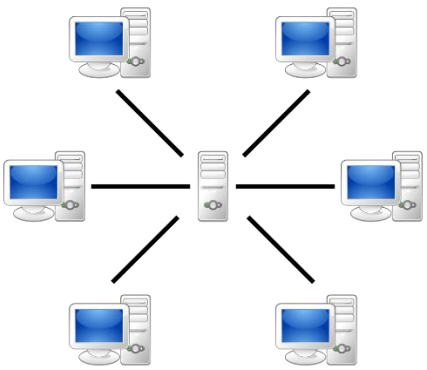

Figura 11 - Xarxa Client/Servidor

- **Predicció del Client:** és una millora del model anterior, que permet disminuir la latència de les accions que realitza el jugador sobre el seu propi personatge. El fonament és que el client executarà codi localment per a controlar el personatge, evitant l'espera d'enviament i resposta del servidor (només per a accions que impliquin un canvi d'estat en el personatge controlat). Ara bé, el servidor pot efectuar correccions sobre l'estat del jugador en el moment en què hi ha un factor extern que li alteri l'estat. Així es manté una coherència entre la instància de joc del client i la del servidor.

2.1 Estudi de Mercat

Com a competidors directes, jocs que mesclin el factor cartes i *battle royale*, no en trobem en el mercat actual. El que sí trobem, però, són grans títols reconeguts mundialment dels dos gèneres per separat. Veiem-ne els referents actuals:

Card Games

- ***Hearthstone* (2014):** joc de cartes col·leccionables digital ambientat en l'univers de *Warcraft*. Enfrontaments 1 contra 1 on cada jugador té una baralla de cartes. Aquestes cartes tenen efectes i habilitats que els jugadors poden usar amb l'objectiu d'eliminar al seu oponent. Cada jugador compta amb 30 vides inicialment, i el primer que arribi a 0 perd. Les cartes tenen un cost i poden ser de diferents tipus: criatures, encantaments, conjurs, secrets...
    - **Desenvolupador:** *Blizzard Entertainment*
    - **Motor usat:** *Unity*

Figura 12 - Partida de Hearthstone

- ***Clash Royale* (2016):** el factor diferencial d'aquest joc respecte aquest gènere és que no consta de torns, sinó que és a temps real. Els jugadors poden llençar les cartes de la seva baralla quan vulguin (i puguin). Ens trobem en un Camp de batalla on l'objectiu és destruir la torre del rei enemiga, protegida per dues torres sentinelles. Així doncs, els jugadors llançaran les seves tropes en llovs estratègics del mapa per a poder arribar i destruir ràpidament les defenses enemigues.
    - **Desenvolupador:** *Supercell*
    - **Motor usat:** motor propi.

Figura 13 - Partida de Clash Royale

Battle Royales

- ***PUBG* (2017):** videojoc multijugador massiu on 100 jugadors es troben en una illa amb l'objectiu de sobreviure i ser els últims en vida. Per a aconseguir aquest objectiu, hauran d'explorar el mapa per trobar bon equipament i armes i així aconseguir avantatge respecte els seus oponents. Hi ha diferents modes de joc: solitari, per parelles o per equips.
    - **Desenvolupador:** *BlueHole Studio*
    - **Motor usat:** Unreal Engine.

Figura 14 - Player Unknown's Battlegrounds

- ***Fortnite: Battle Royale* (2017):** al igual que en el PUBG, 100 jugadors són llençats en una illa i han de sobreviure per quedar els últims en peu. És innovador pel seu estil *cartoon* i una mecànica diferencial: la construcció. Els jugadors han de recol·lectar recursos (fusta, ferro...) per poder construir les seves pròpies cobertures i poder acabar amb els seus contrincants. També hi ha diferents modes de joc, així com modes especials durant períodes de temps limitats.
    - **Desenvolupador:** *Epic Games*
    - **Motor usat:** *Unreal Engine*

Figura 15 - Fortnite: Battle Royale

També podem destacar aquells jocs que han donat un toc innovador i diferent al gènere, com podrien ser:

- ***Tetris 99* (2019):** multijugador online on 99 jugadors juguen al famós *Tetris* simultàniament. Cada línia que eliminen del seu taulell serveix d'atac contra altres jugadors (ja que aquestes línies ompliran el taulell dels rivals, dificultant la seva partida). L'objectiu és aconseguir repel·lir els atacs dels oponents i mantenir-se en vida fins al final. El joc acaba amb un guanyador, que serà l'últim supervivent.
    - **Desenvolupador:** *Arika*
    - **Motor usat:** motor propi

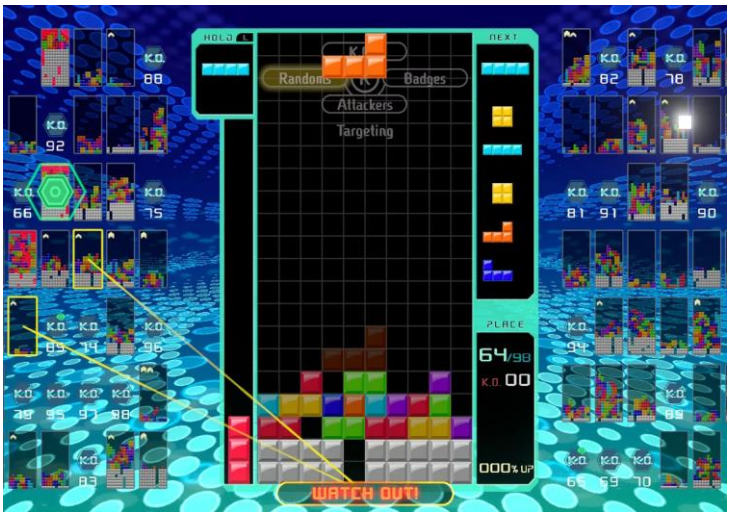

Figura 16 - Partida Tetris 99

- ***SpellBreak* (en versió *Alpha*):** agafa el gènere del *battle royale* i li aporta un toc fantàstic i de RPG, ja que els jugadors són mags de diferents classes i compten amb habilitats i encanteris màgics de diferents elements. Durant la partida podran anar equipant-se amb elements màgics més poderosos i orientar la seva estratègia en funció de les habilitats que van adquirint (defensives, ofensives...). Aquesta jugabilitat veiem que encaixa molt bé amb l'ambientació medieval que han adoptat.
    - **Desenvolupador:** *Proletariat*
    - **Motor usat:** *Unreal Engine*

Figura 17 - SpellBreak

3. METODOLOGIA

Hem decidit adoptar la metodologia clàssica de desenvolupament de videojocs: **PREPRODUCCIÓ**, **PRODUCCIÓ** i **POSTPRODUCCIÓ.**

La **preproducció** consta bàsicament de la cerca d'informació rellevant per a l'elaboració del projecte i preparació de les eines necessàries per a donar inici a la següent fase (creació del repositori, projecte de *Unity*...). Aquesta part també és important pel fet de ser l'etapa on desenvolupem i detallem la idea del joc, així com l'organització del projecte. En el nostre projecte, aquesta fase engloba les següents sub-fases:

1. ***Concept Discovery*:** tenir la mínima base del prototip, experimentar i comprovar les millors alternatives i procediments per a fer créixer el codi de la manera més òptima i clara.
2. ***Vertical Slice*:** tenir la funcionalitat bàsica de connexió a Internet és primordial en aquesta fase. També implementar una primera versió de la UI in-game, així com les primeres cartes per a començar a testejar el *core gameplay* del que seria una partida normal.

La fase de **producció,** per a nosaltres, constarà de les següents dues sub-fases:

3. ***Alpha*:** implementar el lobby principal i tota la funcionalitat de començar la partida, així com els esdeveniment especials del mapa (zona segura*, chests, airdrops...*). En aquesta fase ja tindrem totes les cartes funcionals, amb les seves habilitats específiques, així com la diferenciació dels tres personatges (*warrior*, *rogue* i *wizard*).
4. ***Beta*:** acabar d'implementar tota la funcionalitat del joc**,** tests interns/externs, *bug fixing* I pulir l'acabat del joc.

La **postproducció** bàsicament per a nosaltres representa la finalització del prototip, ja que el nostre objectiu inicial és aquest: desenvolupar un prototip d'un videojoc. El nostre objectiu final no és anar més enllà i seguir un manteniment i actualitzacions típiques dels jocs actuals. Aquesta fase la nostra última fase de desenvolupament:

5. ***Gold*:** aplicar els retocs finals per a deixar el prototip sòlid, estable i funcional.

(Paral·lelament, al llarg de la realització d'aquestes fases també es van desenvolupant els diferents apartats de la rúbrica.)

Pel fet que el nostre objectiu final es desenvolupar un prototip, volem adoptar certs aspectes del mètode de Mark Cerny ("*The Method*"), un dels dissenyadors de videojocs de més prestigi a nivell mundial. Aquests aspectes ens permetran desenvolupar una millor versió final del prototip, i aquests punts clau són**:**

1) **Realitzar una pre-producció intensiva.**
   - Fer èmfasi en les 3C (*Character, Camera, Controls*).
   - Repetir allò divertit i descartar allò avorrit.
2) **Importància de la primera versió jugable.**
   - Centrar-nos pulir les mecàniques implementades.
   - Treballar-lo fins tenir la qualitat i solidesa suficient
3) **Evolució del disseny del joc**
   - Iterar molt inicialment, i anar reduint la quantitat de canvis al llarg del projecte.

Per a la realització de les tasques contemplades en cada una de les sub-fases de la producció, treballaré amb la metodologia *Agile* anomenada ***Kanban***, on cada tasca va avançant per quatre fases diferents que hem establert:

1. ***Planned*****:** tasques pendents a realitzar.
2. ***In Progress*****:** tasques que actualment estan sent realitzades.
3. ***Testing*****:** un cop finalitzades, passen a l'estat de revisió.
4. ***Complete*****:** tasques revisades i completades.

4. GESTIÓ DEL PROJECTE

4.1 Procediment i Eines per al seguiment del projecte

4.1.1 GANTT

A través de la plataforma *TeamGantt* he elaborat el diagrama amb totes les tasques que he considerat necessàries i rellevants per al desenvolupament i finalització del projecte.

Amb aquesta eina podem observar el temps assignat a cada tasca, així com el seu progrés. Aquí podem veure cadascuna de les tasques, agrupades en fases (explicades en l'apartat de metodologia).

1. ***Concept Discovery***

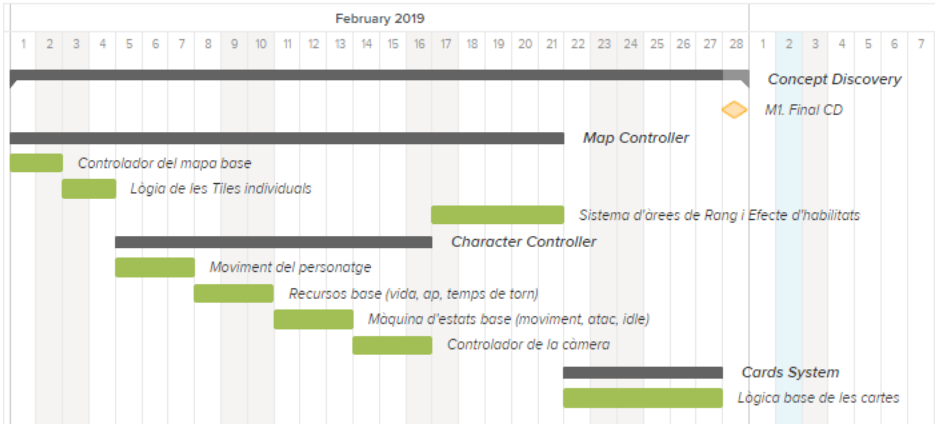

Figura 18 - Concept Discovery

2. ***Vertical Slice***

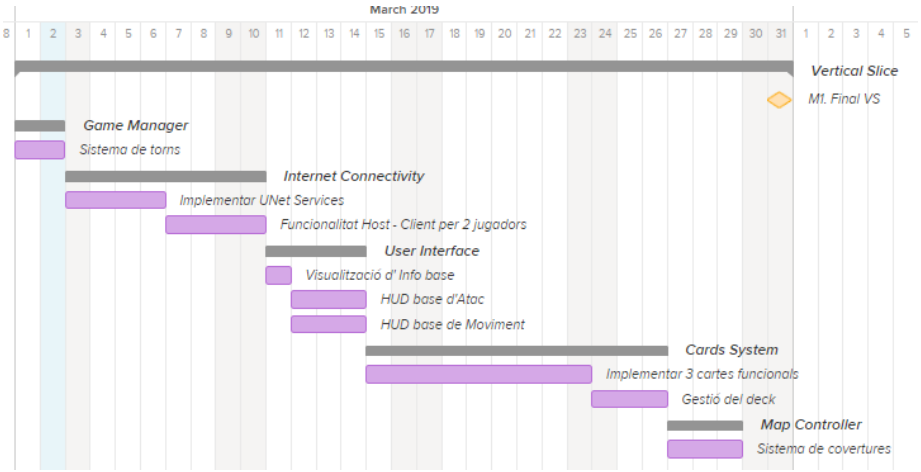

Figura 19 - Vertical Slice

3. *Alpha*

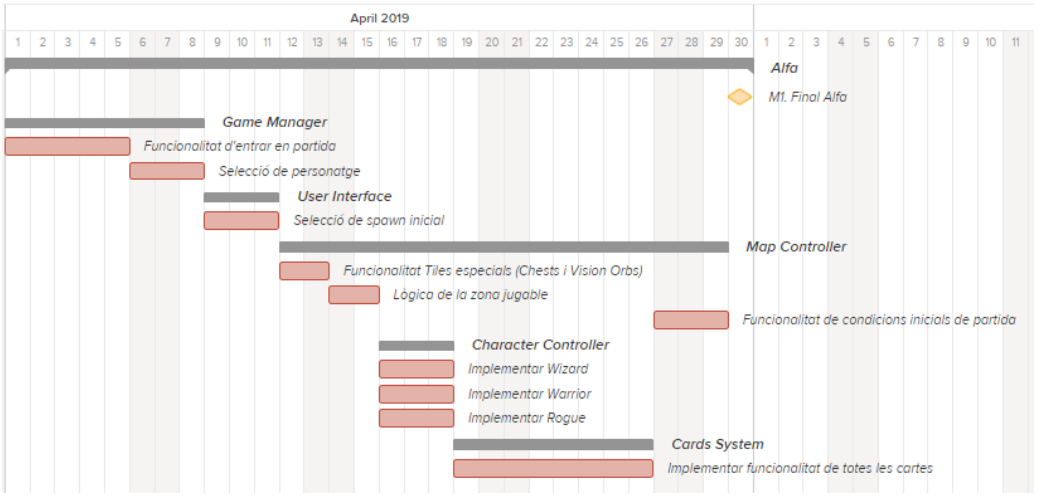

Figura 20 – Alpha

4. *Beta*

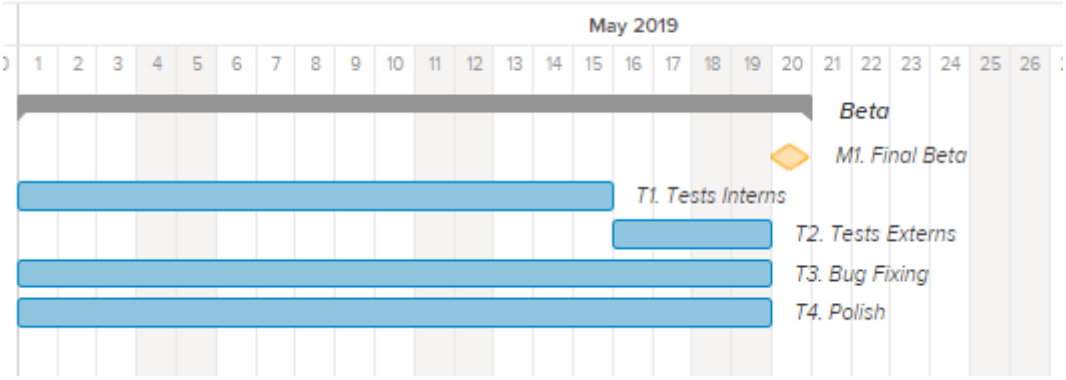

Figura 21 – Beta

5. *Gold*

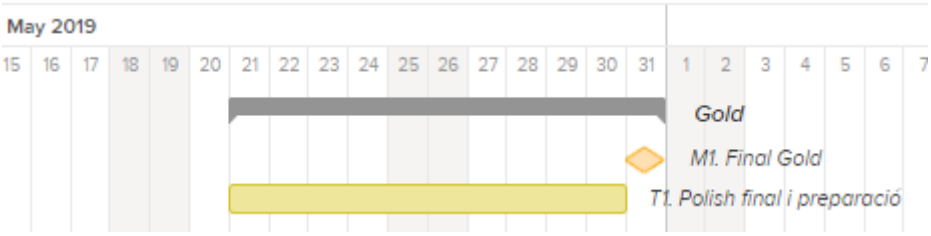

Figura 22 - Gold

A través d'aquest link podem visualitzar el diagrama complet, així com el progrés de cada una de les tasques.

3.1.2 HacknPlan

Hem creat un projecte en la plataforma *HacknPlan* per anar controlant les tasques que anem realitzant en cada una de les *milestones* proposades en el diagrama de Gantt i tenir una monitorització del procés de desenvolupament.

Una tasca està composta per diferents apartats que la classifiquen i defineixen:

- **Títol.**
- **Categoria.**
- **Descripció:** es defineix l'estat ideal en el que s'hauria d'arribar un cop finalitzada la tasca i, a mesura que es treballa en ella, es van indicant els canvis i implementacions en aquest apartat.
- **Etiquetes:** per classificar-la dins d'una o més àrees del projecte (p.e: *Card System, Map Controller, User Interface...*)
- **Usuaris:** encarregat/s de realitzar la tasca.

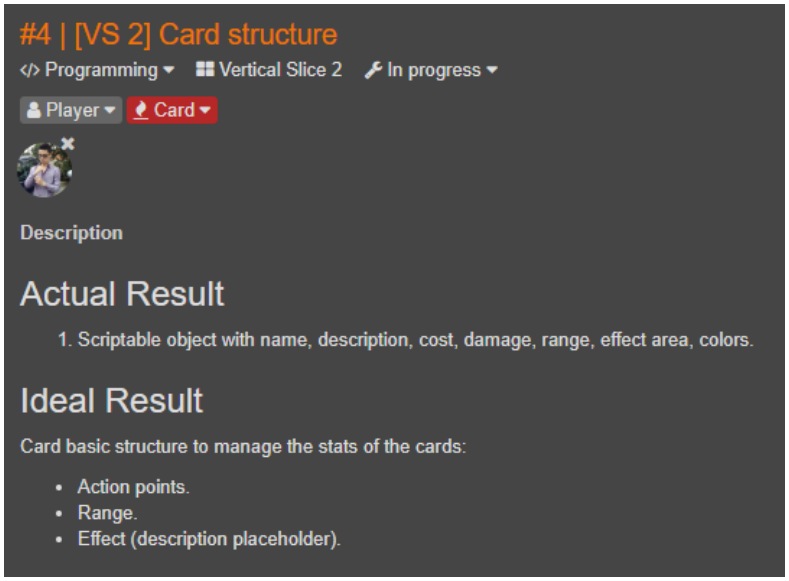

Figura 23 – Exemple d'una Tasca

Tenim un taulell dividit per columnes on col·loquem cada una de les tasques depenent de l'estat en el que es troba actualment, seguint la metodologia *Kanban* explicada en l'apartat anterior.

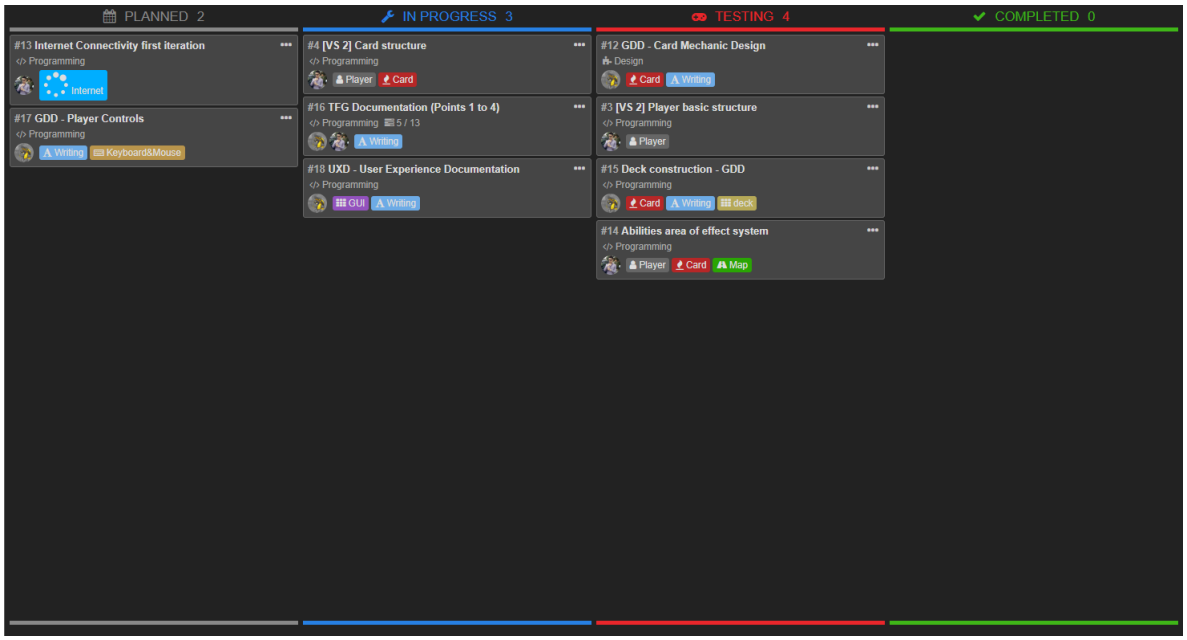

Figura 24 - Taulell de HacknPlan

4.1.3 GitHub repositori en xarxes, Git eines de control de versions

Hem creat un projecte privat en la plataforma GitHub, ja que hem comprat una sèrie d'*assets* de pagament que no podem compartir públicament.

4.2 Eines de validació

Per anar validant l'estat i qualitat del prototip al llarg del procés de desenvolupament realitzarem dos tipus de tests:

- **Tests interns:** cada cop que es desenvolupi una nova versió jugable del joc (que contingui noves característiques) realitzarem internament proves per a trobar la màxima quantitat d'errors possibles i arregrar-los en la mesura del possible.
- **Tests externs:** en les sub-fases Beta i Gold realitzarem tests amb jugadors externs al projecte, que provaran el joc durant un període determinat de temps. Passat aquest període se li passarà un formulari de *Google Forms* que hauran d'omplir segons la seva opinió i sensacions després d'haver jugat al joc. Aquests formularis ens indicaran quines característiques estan funcionant i quines no, i podrem procedir a realitzar les correccions adients.

4.3. DAFO

Si volguéssim que aquest projecte fos llençat al mercat com un videojoc real hauríem d'analitzar-lo detalladament tant de forma interna com externa, per veure la situació en la que ens trobem i valorar-la objectivament, sabent així abast i viabilitat del producte.

Així doncs, realitzem un DAFO i observem els punts forts i febles de la nostra proposta:

	Positius	**Negatius**
Origen Intern	**Fortaleses** - Innovació en *gameplay* i mescla de gèneres. - Coneixement del trets clau dels gèneres per exprimir-los al màxim. - Corba d'aprenentatge suau (menys frustració per al jugador).	**Debilitats** - Temps de producció bastant limitat. - Apartat artístic poc treballat. - Contingut limitat (inicialment, sense sistema de progressió).
Origen Extern	**Oportunitats** - Sector del mercat específic poc explotat. - Popularitat i atracció per a aquest tipus de jocs. - Barrera d'entrada inicial baixa (seria un joc gratuït).	**Amenaces** - Grans competidors en el mercat (*Fortnite*, *Hearthstone*...). - Sobresaturació de jocs en el mercat actual. - Connexió en línia limitada (sistema de *Hosting*).

Taula 1 - DAFO

4.4. Riscos i pla de contingències

A continuació plantegem un llistat de riscos que poden sorgir al llarg del projecte, i les mesures preses en cas que aquests apareguin, per tal de reconduir la situació i seguir endavant amb el projecte.

Risc	**Solució**
Impossibilitat d'implementar animacions dels personatges.	Dissenyar els personatges com a peces d'escac estàtiques, sense animacions.
Complexitat a l'hora de programar la funcionalitat de cada una de les cartes.	Reduir el nombre de cartes diferents a aquelles cartes més importants i principals, i reaprofitar funcionalitats.
Tasques inacabades al final de la *milestone*.	Passar-les a la següent *milestone* i reduir la càrrega/pes de les tasques de la *milestone* actual (prioritzant les més importants)
La funcionalitat Online no es pot implementar (per limitacions de codi, complexitat massa elevada...)	El joc passaria a jugar-se en mode local, en una mateixa màquina.
La visió general del joc difereix entre nosaltres (en Sergio i jo), provocant discordances i malentesos en l'elaboració de les tasques.	Cada setmana es realitzen reunions on cadascú de nosaltres exposa en el que està treballant i informa de l'estat actual del projecte.
La Interfície d'Usuari plantejada és massa complexa (en quant a animacions, interaccions, etc.) d'implementar.	Es planteja una versió reduïda i simplificada amb els elements principals bàsics necessaris.

Taula 2 - Llistat de Riscos i contingències

4.5. Anàlisi inicial de costos

El nostre objectiu, com he comentat en anteriors ocasions, no és desenvolupar un videojoc complert per a llençar-lo al mercat, sinó realitzar un PROTOTIP de videojoc, amb potencial per ser desenvolupat i ampliat per arribar a ser un joc complert.

En el cas real, els únics costos que hem assumit específicament per a la realització d'aquest projecte en han sigut els dels *assets* i paquets d'art i models 3D adquirits a través de la *Asset Store* i la plataforma *Humble Bundle*:

Element	**Cost**
Art Fantasy RPG Pack	18,00 €
UI Fantasy RPG Pack	18,00 €
Polygon: Adventure Pack	14,29 €
Polygon: Fantasy Characters	9,28 €
Total	**57,57 €**

Taula 3 - Costos del projecte

El que podem plantejar a continuació és un **escenari hipotètic** on es contemplin els costos del procés de desenvolupament que suposaria el projecte considerant les següents suposicions:

- El temps de desenvolupament és de 12 mesos (Gener 2019 – Desembre 2019).
- L'equip de desenvolupament està format per en Sergio Sáez i jo, Jordi Oña com a treballadors (programador i dissenyador, respectivament) que cobren un sou mensual.
- El model de negoci adoptat per al videojoc és el model *Free to Play* (amb *In-App Purchases*).

Així doncs, hem realitzat un Anàlisi de Costos que consta d'una fulla de Costos, una fulla d'Ingressos, la fulla de Balanç i la de Depreciació del Material usat.

Observem que aquest projecte costaria al voltant d'uns **40 000 €.**

5. DESENVOLUPAMENT DEL PROJECTE

5.1. Introducció

Seguint la metodologia mencionada en l'Apartat 3, aquest Prototip de Videojoc el realitzem conjuntament en Sergio Sáez i jo, Jordi Oña.

El meu treball consisteix en implementar en codi tota aquella funcionalitat dissenyada pel meu company.

5.2. Tecnologia usada

Per al desenvolupament del prototip utilitzem el motor explicat anteriorment, **Unity**. Aquest motor em permetrà desenvolupar tot el codi de cada un dels sistemes a implementar.

Complementàriament incorporem el Servei ***Multiplayer*** de Unity (UNET) que habilita la connexió en línia entre jugadors.

Per a comprendre millor el procés de desenvolupament del projecte cal introduir primer tota aquella terminologia i elements base que utilitzo dins el motor:

- **Escena:** un joc està format per diverses escenes, que són els "nivells" on es col·locaran totes les entitats (menús, decoracions, càmera, personatges...) que el faran funcionar.
- **GameObject:** és la classe base de totes les entitats que es troben dins les escenes.
- **Component:** classe base de tots els elements que podem implementar en un *GameObject* per a donar-li una certa funcionalitat.
- **Sistema de Prefabs:** sistema que ens permet configurar, carregar i guardar un *GameObject* amb tots els seus components, valors i objectes fills com a *assets* re-usables. Aquests són guardats dins el projecte i poden ser carregats a l'escena desitjada.
- **Script:** component que conté codi en llenguatge C# que s'encarregarà d'executar la funcionalitat programada.
- **Collider:** component per a detectar col.lisions entre *GameObjects*

- **ScriptableObjects:** bàsicament són assets guardats en el projecte que contenen informació, usats per a serialitzar objectes i poden ser instanciats dins una escena.

Figura 25 - Exemple d'Escena de Unity

5.3. Programació dels sistemes

En aquest apartat definiré cada un dels sistemes principals implementats en el prototip actualment, la seva funció, el procés de desenvolupament i, en alguns casos, possibles alternatives per a la seva implementació.

5.3.1. Sistemes Generals

He decidit seguir el patró *Singleton* per a integrar els tres sistemes que considero "globals". Aquests són el *GameManager*, *el MapController* i el *UIController*. Aquest patró consisteix en generar una única instància de cada un d'aquests sistemes dins el joc i generar un punt global d'accés a ells. D'aquesta manera evitem que per qualsevol motiu es dupliqui un d'aquests elements i impossibiliti el funcionament correcte del joc.

1) ***GameManager*:**

   Aquest sistema és el que s'encarrega de controlar el flux del joc, de inici a fi de la partida.

   Les funcionalitats que he implementat són:

- Control de l'estat de la partida: es respon als canvis d'estat amb la funcionalitat desitjada (p.e: inicia partida quan tots els jugadors estiguin llestos, finalitza partida quan quedi un únic jugador viu...).
- Controlar els torns dels jugadors: per saber en tot moment el jugador actiu (el que està realitzant el torn), controlar els temps (de partida i de torn) i donar inici i fi a cada un dels torns dels jugadors.
- Donar accés a la resta de sistemes generals del joc: conté les referències del *UIController i MapController* per a que elements externs puguin accedir a les seves funcionalitats.

```
#region Variables

// GameManager Setup notification
public static bool isReady = false;

// Managers
private static GameManager instance = null;
private MapController mapController = null;
private UIController uiController = null;
private EventManager eventManager = null;

// Players Management
public List<PlayerController> players = new List<PlayerController>();
private PlayerController activePlayer = null;
public PlayerController localPlayer = null;
public Color highlightedColor = Color.green;
```

Figura 26- Variables controlades pel GameManager

```
// Enums
#region Enums

public enum GameState
{
    MATCH_INIT = 0,
    MATCH_SPAWN,
    MATCH_START,
    MATCH_IN_PROGRESS,
    MATCH_FINISHED
}

#endregion
```

Figura 27 - Estats de la partida

2) *MapController:*

En un joc d'estratègia per torns on els jugadors es mouen per un mapa dividit en caselles (*Tiles*), és molt important tenir un control sobre què està passant en cada una de les caselles del mapa.

El tractarem com una graella 2D on l'origen de coordenades (0, 0) es troba en la cantonada inferior esquerra. Tot i ser un joc amb models 3D, les mecàniques implementades només contemplaran dues dimensions (X, Y), per tant, crec que és l'aproximació òptima per a tractar tota la funcionalitat del mapa.

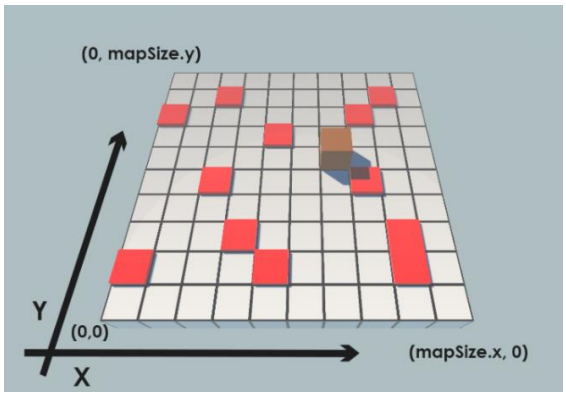

Figura 28 - Coordenades del Mapa

La unitat mínima del mapa serà la ***Tile***, un script associat a cada casella que permetrà tenir control sobre:

- La posició (x, y) de la casella.
- El color: quan el jugador es prepari per alguna acció en concret sobre una casella, aquesta podrà canviar de color en funció de l'acció (atac, moviment, etc.).
- L'ocupació: en el moment en què un obstacle o un jugador se situï sobre una *Tile* concreta, aquesta prendrà consciència de l'element en qüestió per a què els altres jugadors puguin interactuar amb ell.
- Movilitat: a través del sistema de *tags* de Unity, indicarem si la casella es *Walkable* o *No Walkable* per a possibilitar (o no) l'accés als jugadors.

Aquestes *Tiles* seran emmagatzemades en una matriu 2D en el *MapController* per tal d'accedir i tractar amb elles des de *scripts* externs.

Aquest sistema també s'encarregarà de:

- Generar les **àrees** de **moviment** del jugador i les àrees de **rang** i **abast** de les habilitats de les cartes.
- Generar els ***paths*** de **moviment** del jugador a partir de la posició inicial i final.

Aquestes funcionalitats les detallarem en l'apartat de Controlador del Jugador més endavant.

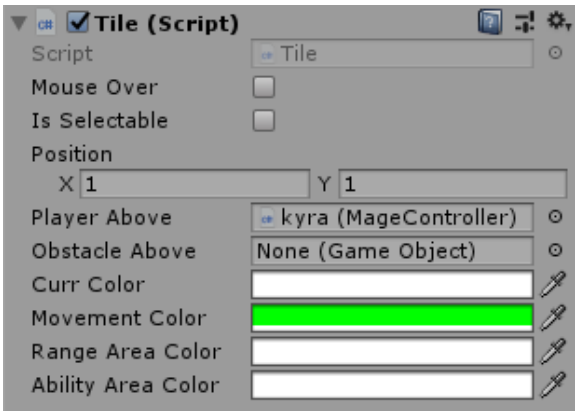

Figura 29 - Component Tile

3) *UIController:*

Aquest sistema és el que permetrà al jugador interactuar amb el joc. A través de l'Interfície d'Usuari es podrà:

- Accedir als menús del joc: pausa, opcions, pantalla de victòria...
- Triar l'acció a realitzar (Atacar, Moure's, etc.).
- Escollir la carta a utilitzar.
- Veure la informació de la partida: temps de torn, temps de partida, jugadors en vida.
- Veure la informació del personatge: vida, escut, punts d'habilitat (AP), jugadors eliminats.

Aquest sistema té les referències de cadascun dels elements visuals (imatges, textos, *sliders*) per a poder modificar-los correctament en funció de l'acció que realitzi el jugador.

Per a implementar correctament la funcionalitat del jugador, aquest controlador consta de quatre estats diferents. Cadascun d'ells correspon a un estat del torn en el que el jugador actiu podrà fer unes accions o unes altres:

- ***IDLE***: estat inicial de la UI. A través d'aquest es pot accedir a qualsevol dels altres.
- ***PAUSE***: estat de la UI quan el jugador obre el menú d'opcions. Es restringeix tota la funcionalitat de joc (moviment i habilitats).
- ***MOVE***: estat en que el jugador es prepara per a moure's, es restringeix la funcionalitat d'usar cartes.

- ***ATTACK***: es restringeix la funcionalitat de moviment per a habilitar tota la funcionalitat de l'ús de les cartes.

Aquest controlador també s'encarregarà de detectar els inputs del jugador per realitzar les accions desitjades.

```
void Update ()
{
    /* --------------------- INPUT DETECTION --------------------- */
    if (Input.GetKeyDown(KeyCode.Escape))
        TogglePause();

    if (state == UIState.PAUSE)
        return;

    if (GameManager.Instance.localPlayer.state >= PlayerController.State.IDLE &&
        GameManager.Instance.localPlayer.state <= PlayerController.State.PREPARE_ATTACK)
    {
        // IDLE
        if (Input.GetMouseButtonDown(1))
            SetIdleMode();

        // SHOW MOVEMENT
        if (Input.GetKeyDown(KeyCode.Space))
            SetMoveMode();

        // PREPARE ATTACK
        if (Input.GetKeyDown(KeyCode.Q))
            SetAttackMode();
    }

    /* ------------------------------------------------------------ */
}
```

Figura 30 - Detecció d'inputs

5.3.2. Controlador del Personatge

El *PlayerController* serà el *script* base que controlarà totes les accions i variables del personatge. En el joc tindrem 3 personatges diferents per escollir: *Mage, Rogue i Warrior*, així que he decidit agrupar totes les característiques base en el script *PlayerController* que actuarà de classe pare dels tres personatges fills: *MageController*, *WarriorController*, i *RogueController*. Aquests tres *scripts* emmagatzemaran la informació única i distintiva de la classe.

Els paràmetres principals d'un personatge són:

- **Vida (HP):** punts de salut. Quan són reduïts a 0, el jugador mor i perd la partida.
- **Punts d'habilitat (AP):** recurs gastat quan un jugador realitza una acció (moure's, usar una carta...). Es recarrega a l'inici de cada torn.
- **Temps de torn:** temps que disposa el jugador per a realitzar les seves accions quan es troba en el seu torn.

Aquest controlador el podem dividir en tres funcionalitats principals:

- **Màquina d'estats (*Finite State Machine*):**

  Per tal de controlar quines accions pot i no pot fer en cada una de les circumstàncies de joc, he decidit implementar una *FSM* que presenta uns estats concrets (*IDLE, MOVE, ATTACK*...). El jugador, en funció de l'estat en el que es trobi, podrà realitzar una acció en concret o una altra. Aquests estats són canviats des de codi, a través del *UIController* mitjançant la detecció d'inputs.

```
switch (state)
{
    case State.SPAWNING:
        ...
    case State.IDLE:
        ...
    case State.PREPARE_MOVE:
        {
            // Mouse collision with tiles
            CheckMovementMouseCollision();

            if (Input.GetMouseButtonDown(0) && targetTile != null && targetTile != currentTile && inTurn)
            {
                if (EnoughAP(movementCost))
                {
                    SpendAP(movementCost);
                    GameManager.MapController.MovePlayerTo(targetTile, this);
                }
                else
                {
                    Debug.Log("You don't have enough AP");
                }
            }
            break;
        }
    case State.WALKING:
        ...
    case State.PREPARE_ATTACK:
        ...
    case State.ATTACKING:
        ...
    case State.SHUFFLING_DECK:
        ...

    default:
        break;
}
```

Figura 31 - Màquina d'estats del PlayerController

- **Sistema de Moviment:**

  Com hem comentat anteriorment, és un joc on el mapa està organitzat per caselles. Per a crear un camí i possibilitar el moviment del jugador d'una casella a una altra he observat 3 algoritmes de *pathfinding* possibles:

  1) *Breadth First Search (BFS)*: partint d'una casella inicial, es consulten les caselles adjacents (nord, sud, est i oest). Per a cada casella veïna es consulten també les seves adjacents, i així fins recórrer tota l'àrea

desitjada. Cada casella té una distància respecte l'anterior i es genera un camí a partir de l'ordre en què s'han consultat.

2) *Dijkstra*: tracta d'explorar els camins més curts des del punt d'origen fins al punt de destí. L'exploració no es realitza de manera uniforme, sinó que anirà processant les caselles de distància més curta fins arribar al destí.
3) *A**: evolució de l'algoritme *Dijkstra*, que no té en compte només la distància de les caselles, sinó que contempla el cost de desplaçament (p.e: algunes caselles podrien tenir propietats especials com fang, dunes.. que dificultessin el camí. Aquestes caselles tindrien un cost major i l'algoritme ho tindria en compte.).

He decidit implementar l'algoritme *BFS*, ja que és l'algoritme més simple i més adequat, segons el meu criteri, a les dimensions del nostre joc: tenim un mapa on només es té compte la distància de les caselles respecte el jugador, no tenim en compte pesos i valors extra, més enllà de saber si la casella és accessible o no.

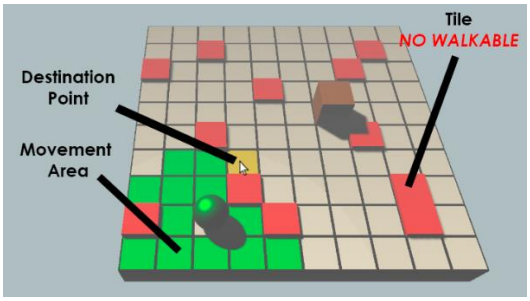

Figura 32 - Àrea de moviment

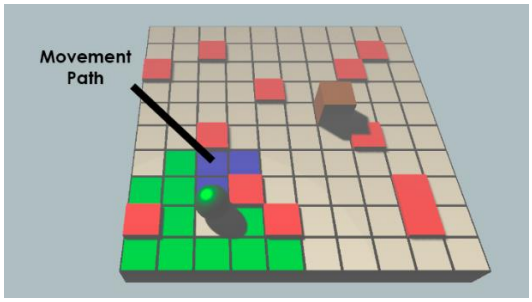

Figura 33 – Camí generat

- **Habilitats:**

  La interacció entre jugadors ve donada principalment per l'ús de les cartes que adquireixen al llarg de la partida. Cada carta, al ser usada, realitza una habilitat, que té una àrea de rang que indica totes les caselles al voltant del jugador a les que es pot usar l'habilitat, i també te una àrea d'efecte, que indica les caselles que quedaran afectades per l'habilitat.

  Aprofito el sistema del mapa que genera les àrees de moviment per a generar aquests dos tipus d'àrees.

Tenim dues llistes de punts:

- ***Range Area:*** llista que guardarà totes les caselles que mostraran l'àrea de rang de l'habilitat (o el moviment del personatge)
- ***Ability Area:*** llista que guarda les caselles que mostraran l'àrea d'efecte de l'habilitat.

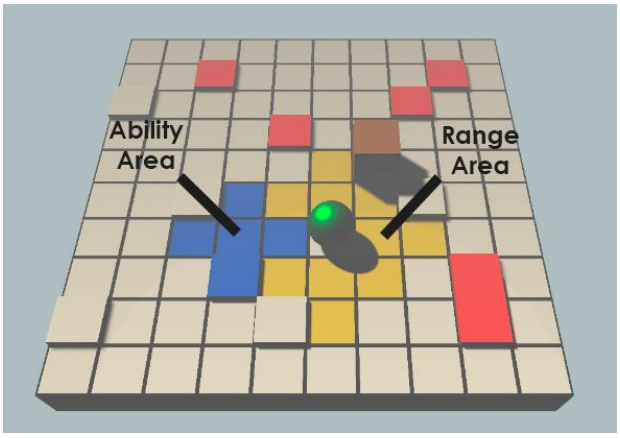

Figura 34 - Àrees d'habilitat

Sistema de Cobertures i Visibilitat:

Implementant el sistema d'habilitats se'm va plantejar un repte: Què passa si es mostra l'àrea de rang d'una habilitat en una zona on el jugador està envoltat d'obstacles? Si l'àrea és suficientment gran com per comprendre les caselles que es troben darrere els obstacles, el jugador no tindria visió d'elles, i per tant, serien caselles inaccessibles.

Per a resoldre aquest problema vaig observar dues possibilitats:

a) Resoldre-ho amb lògica 3D:

1- Llençant rajos des de la posició del jugador cap a tots els obstacles de l'àrea.
2- Duplicar els rajos i situar-los als extrems dels obstacles, generant així àrees de visibilitat.
3- Detectar les caselles dins les àrees de visibilitat i ometre les restants.

b) Resoldre-ho amb lògica 2D: aquest és el mètode que he utilitzat, ja que tinc un major control sobre les posicions de les caselles (que són punts 2D en el mapa) i totes presenten les mateixes dimensions. En el cas de lògica 3D, els

rajos que apunten als obstacles podrien no quadrar amb les dimensions de les caselles, provocant errors de càlcul de les caselles dins el rang de visió. Aquest sistema de visió 2D l'he plantejat de la següent forma:

1- Dividim l'àrea en 8 seccions (octants). Aquestes àrees estaran delimitades per vectors de direcció, situats a cada 45 graus . Per a cada secció, computarem el **FOV** (*Field Of View*). Resoldrem la visió per a l'octant 0.

   (Podem aplicar la mateixa solució a cada octant restant simplement traslladant les coordenades a la secció desitjada).

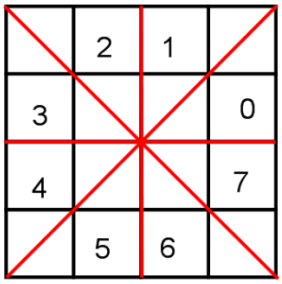

Figura 35 - Divisió d'àrea per octants

2- L'octant 0 està delimitat inicialment pels vectors direcció *TOP* (1,1) i *BOTTOM* (1,0). Bàsicament hem de **comprovar quines caselles es troben compreses entre els vectors *Top* i *Bottom*** generats.

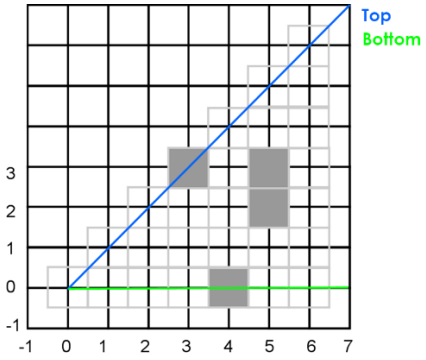

Figura 36 – Pas 1: Generar vectors Top/Bottom

3- Comencem per l'origen de coordenades, que serà la posició del jugador. Comprovarem les caselles per columnes, d'esquerra a dreta i de baix a dalt.

4- Anirem comprovant i marcant l'estat de les caselles en l'ordre corresponent, detectant el pas d'una casella "buida" (sense obstacle al damunt) a una "opaca" (amb obstacle al damunt) i viceversa.
   - Pas de casella BUIDA a OPACA: generem un nou vector *bottom* que passi per l'extrem superior esquerra de la casella opaca.
   - Pas de casella OPACA a BUIDA: generem un nou vector *top* que passi per l'extrem inferior dret de la casella opaca.

5- Un cop processada tota la columna s'actualitza el FOV (els nous vectors generats són vinculats entre ells) i es passa a la següent columna.
   Per a detectar la visibilitat en aquesta nova columna es tindran en compte els nous vectors generats.

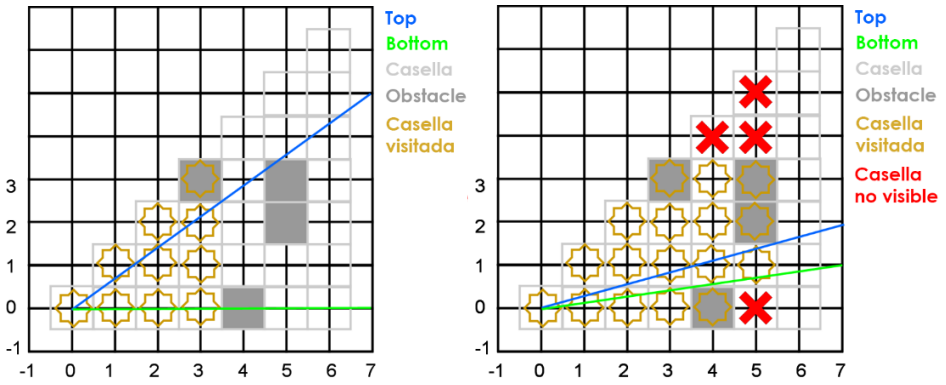

Figura 37 - Computació del FOV pel primer octant

6- Repetim els passos 5 a 7 fins a computar el FOV de cada columna del primer octant.

7- Finalment repetim aquest procés per la resta d'octants i ja tenim el **Sistema de Visió** implementat.

5.3.3. Sistema de Cartes

Aquest és l'últim sistema principal que completa la jugabilitat base del joc. Bàsicament, les cartes són els recursos que el jugador utilitza per a derrotar els seus rivals. Aquestes cartes consumeixen els punts d'habilitat del jugador al ser usades i realitzen una habilitat concreta sobre la seva àrea d'efecte.Podem dividir aquest sistema en tres seccions principals.

1. Card:

Dins aquest sistema, la unitat mínima i principal és el *ScriptableObject* anomenat *Card*. Aquest element conté la informació bàsica de la carta:

- Nom
- Descripció
- Imatge
- Colors (àrea de rang i àrea d'efecte)
- Cost: punts d'habilitat necessaris per usar la carta
- *Attack Damage*: mal de combat que farà si l'habilitat és llençada sobre un enemic
- *Range/Attack area Length*: per a controlar les dimensions de les àrees a generar quan les cartes són seleccionades.

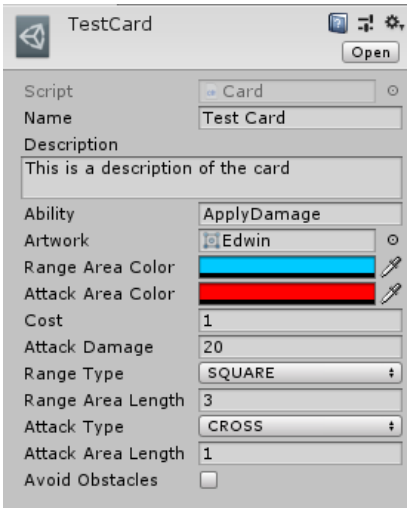

Figura 38 – exemple de Card mostrada en l'ispector

El fet de ser un *ScriptableObject* facilita molt la creació de noves cartes. Només hem d'anar creant fitxers de tipus *Card* i definint la informació mencionada anteriorment.

Aquest element es vincula amb el *MapController* per a generar les àrees de rang i efecte de l'habilitat corresponent a partir:

- Posició del jugador: casella inicial per on l'algoritme es començarà a propagar, generant l'àrea de rang
- Posició del *mouse*: es detecta quina casella està seleccionant el jugador amb el ratolí, i aquesta serà la casella inicial per on l'algoritme es començarà a propagar, generant l'àrea d'efecte.

- Longitud de rang/efecte: determina les dimensions de les àrees generades (les iteracions que realitzarà l'algoritme abans d'aturar la propagació).
- Tipus d'àrea: determina l'algoritme usat per a generar l'àrea.

```
public void ShowRangeArea(Vector2Int tilePos)
{
    GameManager.MapController.ShowRangeArea(tilePos, rangeType, rangeAreaLength, rangeAreaColor, avoidObstacles);
}

public void ShowAbilityArea(Vector2Int tilePos)
{
    GameManager.MapController.ShowAbilityArea(tilePos, attackType, attackAreaLength, effectAreaColor);
}
```

Figura 39 - Vinculació amb el MapController

```
public void ShowRangeArea(Vector2Int position, AreaType areaType, int range, Color tilesColor, bool avoidObstacles)
{
    switch (areaType)
    {
        case AreaType.NORMAL:
            ...
        case AreaType.SINGLE:
            ...
        case AreaType.CROSS:
            ...
        case AreaType.SQUARE:
            {
                StartRangeBFS(position);
                PropagateRangeSquare(range, tilesColor, avoidObstacles);
                break;
            }
        case AreaType.CUSTOM:
            ...
        default:
            break;
    }
}
```

Figura 40 - Generació d'àrees del MapController

```
public void PropagateRangeSquare(int range, Color tilesColor, bool avoidObstacles)
{
    Vector2Int initPoint = Vector2Int.zero;
    Vector2Int actualPoint = Vector2Int.zero;

    initPoint = rangeAreafrontier.Dequeue();
    initPoint.Set(initPoint.x - range, initPoint.y - range);

    for (int i = 0; i <= range * 2; i++)
    {
        for (int j = 0; j <= range * 2; j++)
        {
            actualPoint.Set(initPoint.x + i, initPoint.y + j);

            if(!rangeAreaVisited.Contains(actualPoint) && IsAccessible(actualPoint))
            {
                rangeAreaVisited.Add(actualPoint);
            }
        }
    }

    // Divide the range area by octanes and CHECK VISIBILITY
    //Vector2Int playerPos = GameManager.GetActivePlayer().currentTile.position;
    Vector2Int playerPos = GameManager.Instance.localPlayer.currentTile.position;


    for (int octane = 0; octane < 8; octane++)
    {
        ComputeFOV(range, playerPos, octane);
    }

    DrawTiles(TileType.RANGE, tilesColor);
}
```

Figura 41 - Algoritme de generació d'àrees (tipus SQUARE)

2. CardDisplay:

La *Card* representa la part lògica de la carta. Per a vincular-la amb la part visual he generat un script anomenat *CardDisplay* que s'encarregarà de configurar els textos, imatges i valors de la cartes que es mostraran en la mà del jugador.

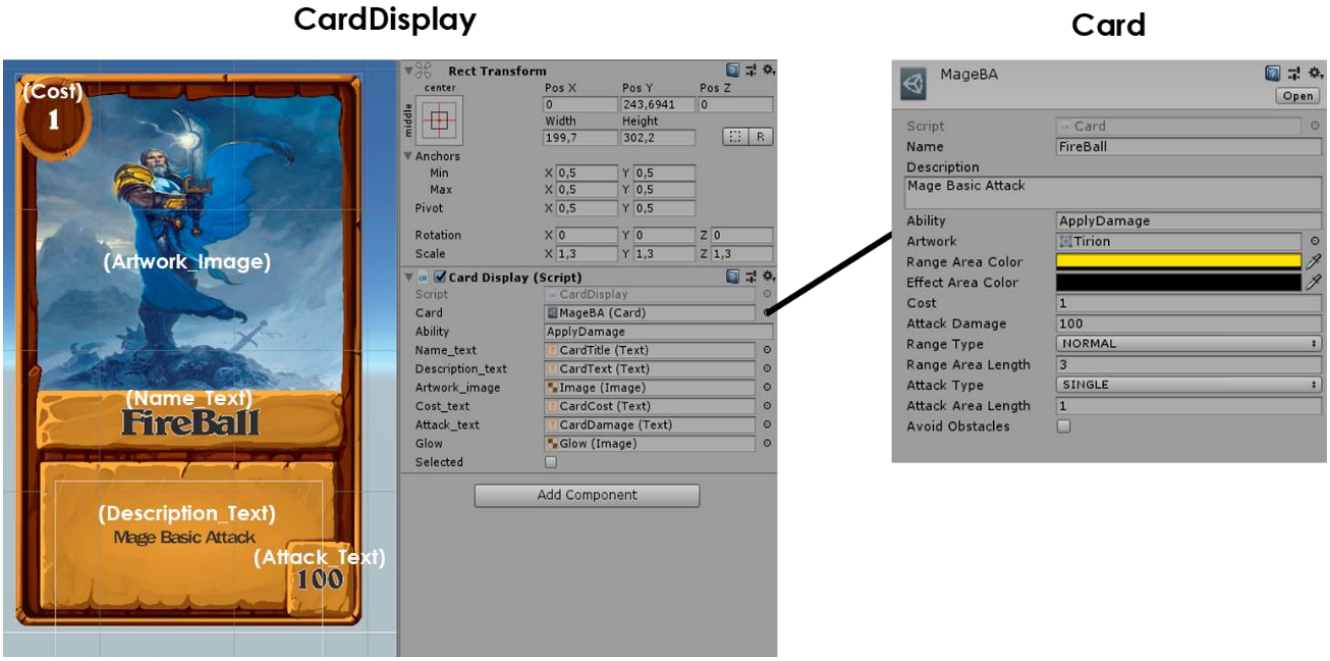

Figura 42 - Vinculació Card/CardDisplay

Aquest script presenta dues funcionalitats principals:

- Visualització de cartes: mitjançant la detecció del ratolí, cada *CardDisplay* comprovarà si aquest es troba sobre. Si és així, la carta es farà més gran i es col·locarà per sobre de la resta per a possibilitar al jugador veure la seva informació amb detall.
- Llançament d'habilitats: tota carta realitza una acció quan és usada. Per a cridar a la funció corresponent en cada carta, el que fem és introduir el nom de la funció que realitzarà l'habilitat desitjada en el *ScriptableObject* Card (en el camp anomenat "*ability*"). Aquest nom és rebut i guardat en el *CardDisplay* sota la variable anomenada també "*ability*". Així doncs, quan una carta és usada, és cridarà a la funció corresponent a partir d'aquest nom.

```
public void SetupCard(Card newCard)
{
    if (card == null)
        return;

    card = newCard;
    ability = card.ability;
    name_text.text = card.name;
    description_text.text = card.description;
    cost_text.text = card.cost.ToString();
    attack_text.text = card.attackDamage.ToString();
    artwork_image.sprite = card.artwork;
}
```

```
public void DoEffect()
{
    // Invoke an Avility Method
    Invoke(ability, 0.0f);
}
```

```
// Abilities
#region Abilities

public void ApplyDamage()
{
    GameManager.MapController.ApplyDamage(card.attackDamage, GameManager.Instance.localPlayer);
}

// ...

#endregion
```

Figura 43 - Procés de llançament d'una habilitat

3. Deck (baralla):

És el script que s'encarrega del control de les cartes i tota la funcionalitat relacionada amb elles. Els elements que gestiona són els següents:

- *handCards*: *array* dels tres cartes (*CardDisplay)* que el jugador veu en pantalla i pot utilitzar al seu torn. Són les cartes de la seva mà.
- *library*: llista de *Cards* que el jugador anirà robant al seu torn.
- *graveyard*: llista de cartes que han sigut usades pel jugador.
- *selectedCard*: carta que el jugador ha seleccionat.
- *cardSlot*: les tres posicions específiques on es generaran els *CardDisplays* robats.

```
// Variables
#region Variables
PlayerController player;
public GameObject cardPrefab;

// Predefined positions to place hand cards
public GameObject[] cardSlot;

// Hand
public CardDisplay selectedCard;
public CardDisplay[] handCard;

// Card management containers
private List<Card> library;
private List<Card> graveyard;

#endregion
```

Figura 44 - Variables del Deck

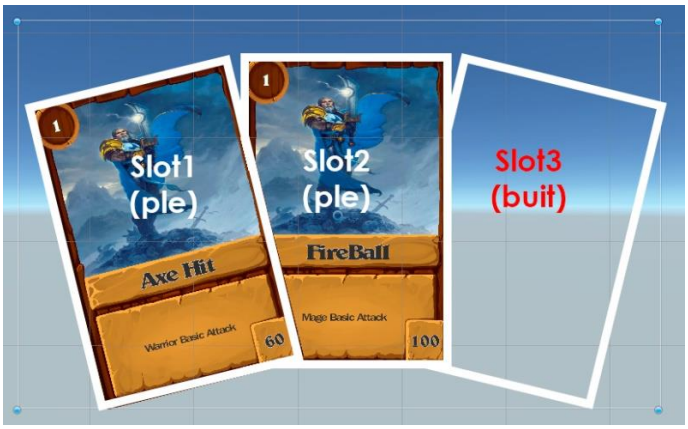

Figura 45 - Mà del jugador

A partir d'aquests elements podem gestionar la funcionalitat següent:

- Generació de la baralla de cartes: quan s'inicia la partida, crea totes les cartes inicials que el jugador pot usar. Les cartes que crea són especifiques i van en funció de la classe del personatge.

```
public void GenerateDeck(PlayerController.Class playerClass)
{
    switch (playerClass)
    {
        case PlayerController.Class.MAGE:
            {
                for (int i = 0; i < 4; i++)
                {
                    Card card2 = Resources.Load("Cards/MageCard1") as Card;
                    library.Add(card2);
                    Card card1 = Resources.Load("Cards/MageCard2") as Card;
                    library.Add(card1);
                    Card card3 = Resources.Load("Cards/MageCard3") as Card;
                    library.Add(card3);
                }
                break;
            }
        case PlayerController.Class.WARRIOR:
            ...
        case PlayerController.Class.ROGUE:
            ...
        default:
            break;
    }
}
```

Figura 46 - Generació de les cartes de la baralla inicial

- Control del el flux de les cartes: s'encarrega de col·locar les cartes a les zones corresponents, en funció de si són:
    - **A) Robades (passen de la biblioteca a la mà)**: va extraient les cartes de la llista *library* situades a l'inici d'aquesta, crea *CardDisplays* amb la informació de la *Card* corresponent i els col·loca als espais buits de l'array *handCard*.

- **B) Usades o descartades (de la mà al cementiri):** el *CardDisplay* és eliminat de l'escena i la carta vinculada en ell (*Card*) és dipositada a la llista *graveyard*.

```
public void DrawCards()
{
    // Check if library is empty to refill with graveyard cards
    if(library.Count == 0)
    {
        library.AddRange(graveyard);
    }

    // Fill the empty card slots of player's hand
    for (int i = 0; i < 3; i++)
    {
        if(cardSlot[i].transform.childCount == 0)
        {
            GameObject newCard = Instantiate(cardPrefab, cardSlot[i].transform);
            Card drawCard = library[0];
            library.RemoveAt(0);

            handCard[i] = newCard.GetComponent<CardDisplay>();
            handCard[i].SetupCard(drawCard);
        }
    }
}
```

```
public void DiscardPlayedCard()
{
    // Put the launched card into the graveyard
    graveyard.Add(selectedCard.card);

    for (int i = 0; i < 3; i++)
    {
        if (handCard[i] == selectedCard)
            handCard[i] = null;
    }

    // Remove the card from the hand
    Destroy(selectedCard.gameObject);
    selectedCard = null;
}
```

Figura 47 - Funcionalitat de robar i descartar cartes

Quan no queden més cartes per robar, totes les cartes del *graveyard* passen de nou a la *library*.

- Funcionalitat "*Deck Options*":
  - **A) *SHUFFLE DECK*:** permet al jugador mesclar tota la baralla i robar noves cartes (les cartes de la mà són dipositades de nou a la llista *library* i, mitjançant un "*shuffle*", aquesta llista es barreja). Un cop barrejada la llista, és donen les noves cartes de l'inici de la llista al jugador.
  - **B) *CHANGE*:** es permet al jugador escollir les cartes desitjades de la mà. Les escollides seran enviades al *graveyard* i es robaran noves cartes de la llista *library*.

- Flux d'ús de cartes:
  1) Al clicar sobre una carta en mode *ATTACK*, el *CardDisplay* notifica al *Deck* que la carta ha sigut seleccionada.

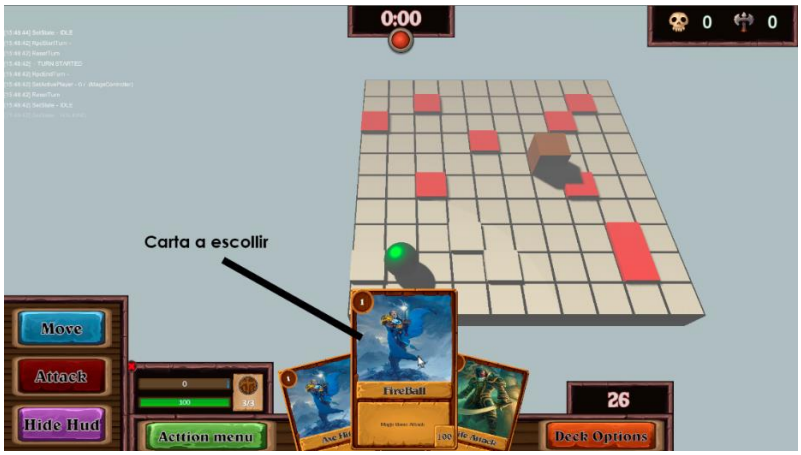

Figura 48 - Pas 1. Selecció de carta

2) Es mostren les àrees de rang i efecte de l'habilitat de la carta i el jugador, si te prou *Ability Points*, pot usar-la.

Figura 49 - Pas 2. Visualització d'àrees

3) El deck cridarà a la *selectedCard* per a que llenci la seva habilitat, i després la descartarà.

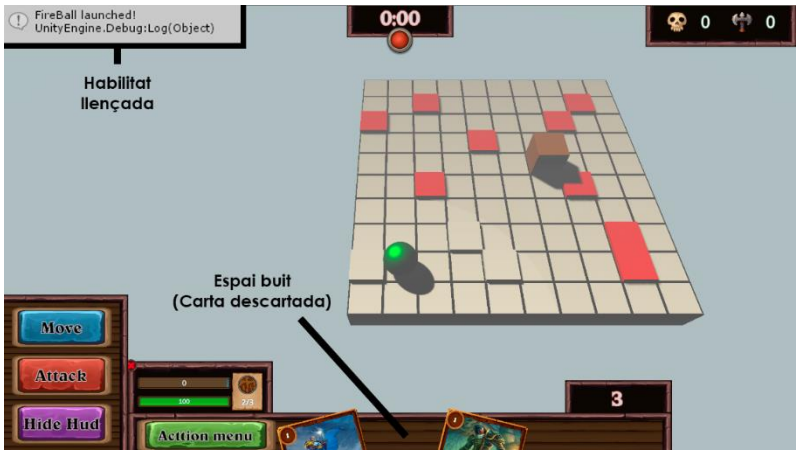

Figura 50 - Pas 3. Ús i descart de la carta

5.3.4. Connectivitat a Internet

Un dels trets principals del nostre prototip és poder jugar en línia. Per a implementar aquesta funcionalitat, Unity compta amb el servei online anomenat *UNet Services*. D'una manera senzilla, dins el projecte es pot activar aquesta funcionalitat que permet connectar als jugadors a través de la xarxa (màxim 20 jugadors per partida).

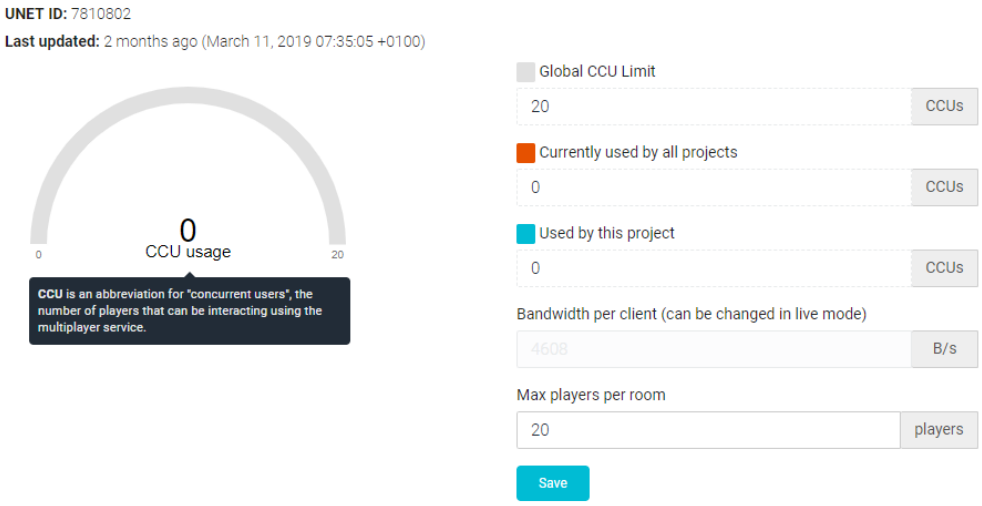

Figura 51 - Panell de control del Servei Multiplayer

Tenim dues vies per a implementar la connectivitat a internet dins el projecte:

1) ***Network Transpot API:*** API de baix nivell que permet construir la teva pròpia infraestructura de xarxes més complexa i específica. Treballa a partir de *sockets*, i permet rebre i enviar missatges representats en paquets de dades (*bytes*).
2) ***High Level Scripting API (HLAPI):*** API d'alt nivell que dona accés a les comandes que cobreixen la gran majoria de funcionalitat dels jocs en línia (comunicació clients-servidor, control de l'estat del joc en línia, sincronització d'elements de joc, creació de servidors...).

He decidit utilitzar la *High Level Scripting API*, ja que és la via més ràpida i simple d'implementar un sistema de *Networking* consistent. A més, aquesta API em dona accés a tota la funcionalitat que necessitem per a desenvolupar aquest projecte.

HIGH LEVEL SCRIPTING API

Amb l'ús d'aquesta API podem establir un protocol en línia Client-Servidor:

- **Servidor:** serà la instància de joc a la qual els altres jugadors es connectaran. Serà qui controlarà l'estat del joc i transmetrà totes les dades als clients.
- **Client:** instància de joc que es connectarà al servidor.

Més concretament, el servidor passarà a ser un ***Host***, i això significa que un dels clients també actuarà com a servidor.

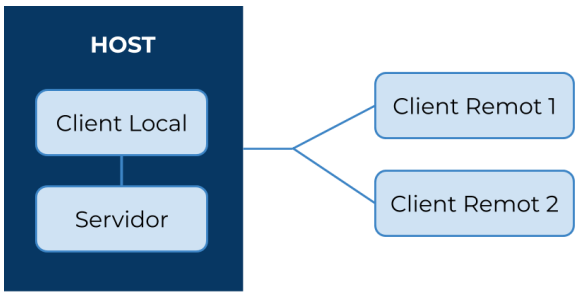

Figura 52 - Diagrama de connexió Clients-Host

La funcionalitat que necessitem implementar per a crear la connectivitat en línia dins el nostre prototip la podem dividir en dues seccions:

- **1. SERVEI "MATCHMAKING"**

  Representa tota la funcionalitat que engloba el tema de creació de servidors (partides disponibles), accés a servidors creats i inici de partida (un cop tots els jugadors hagin entrat al servidor corresponent i estiguin preparats).

  Tota aquesta funcionalitat la he implementat gràcies a un *asset* gratuït de Unity anomenat ***Network Lobby***. Aquest *asset* compta amb una escena pre-creada amb un *LobbyManager*. Aquest serà l'encarregat de portar tota aquesta funcionalitat *Matchmaking* i controlar la xarxa del joc. Hem d'omplir una sèrie de camps específics per a configurar la connectivitat en línia:

  - *Play Scene*: escena de joc que el *LobbyManager* carregarà quan tots els jugadors estiguin preparats per començar la partida.
  - *Game Player Prefab*: *Prefab* que carregarà per a cada jugador quan aquest entri a la partida.

- *Spawnable Prefabs*: tots aquells *Prefabs* que poden ser creats i instanciats dins la escena de joc online.

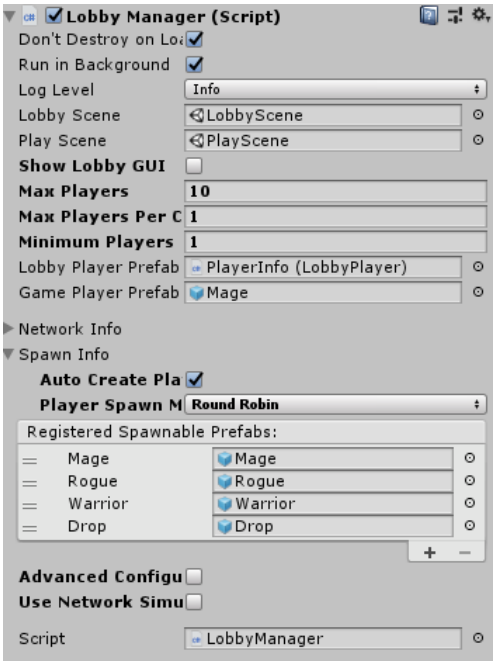

Figura 53 - Lobby Manager

Bàsicament, aquest *asset* compta amb una sèrie de pantalles i menús per on el jugador pot navegar i se li presenta tota la funcionalitat mencionada anteriorment.

Flux d'inici del joc:

1) El jugador inicia el joc i se li mostra un menú inicial on escull un nom pel seu personatge i la classe.

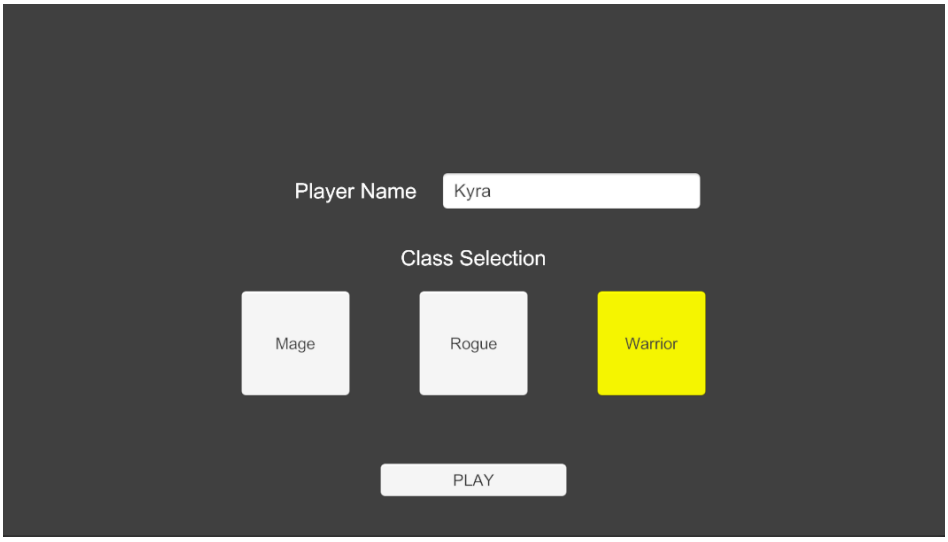

Figura 54 - Menú inicial del joc

2) Dins el *lobby*, pot crear una partida o buscar-la d'entre la llista de servidors. També pot crear la partida amb la opció "*Play and Host*" (aleshores, els jugadors que vulguin entrar a la seva partida ho faran introduint la IP específica).

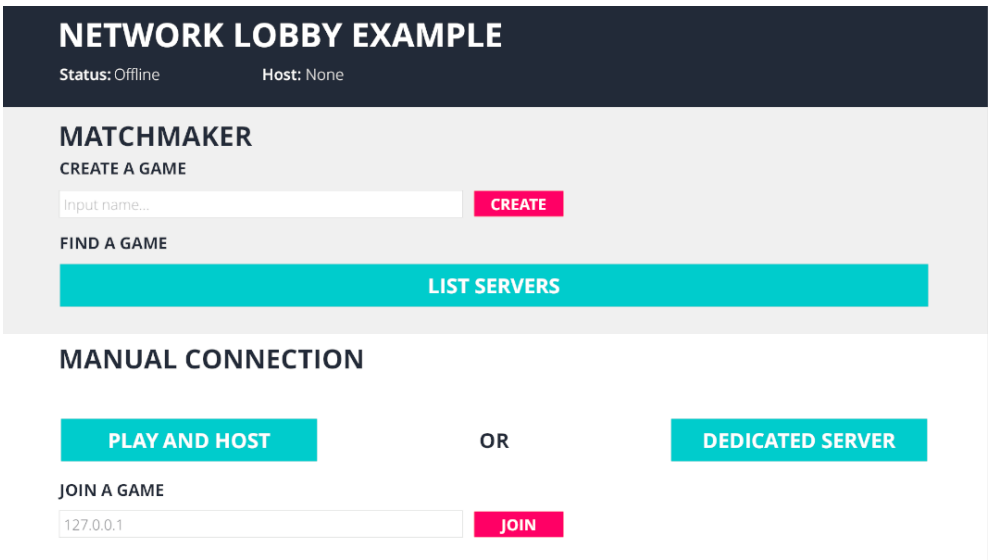

Figura 55 - Pantalla inicial del Lobby

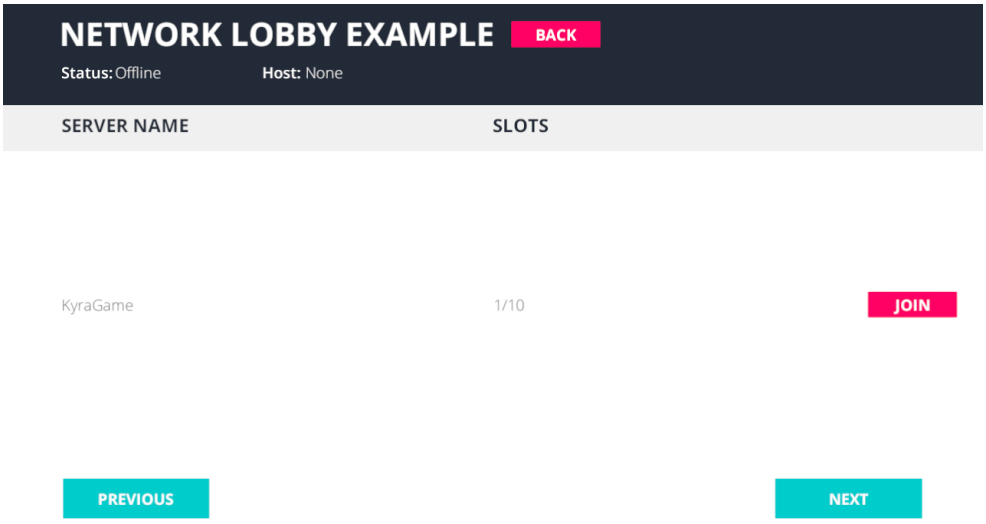

Figura 56 - Llista de partides disponibles

3) Un cop dins la sala, veurem tots els jugadors que disputaran la partida, i per a començar-la, hauran de prémer el botó "JOIN"

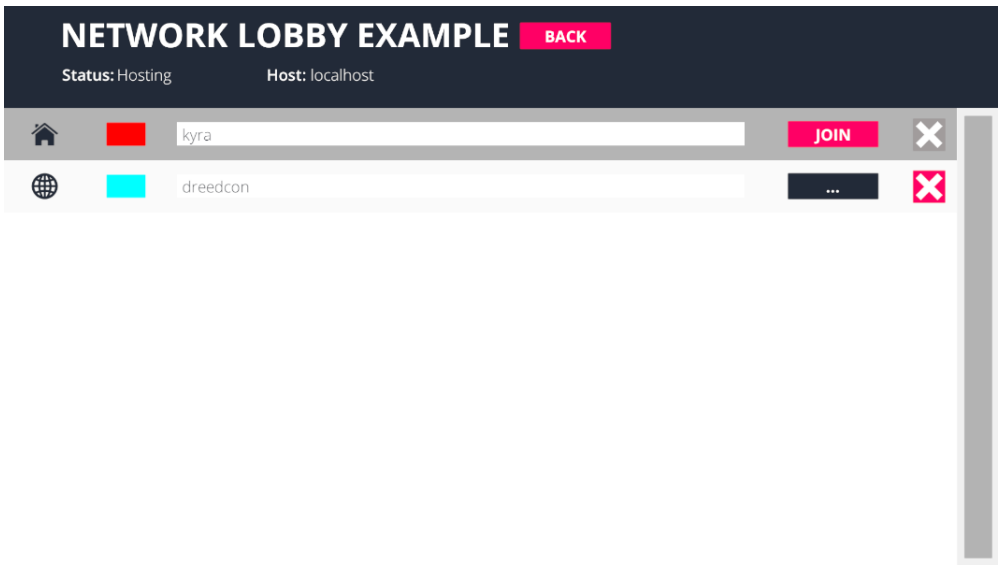

Figura 57 - Menú de la sala

- **2. SINCRONITZACIÓ DE LA PARTIDA EN LÍNIA**

Aquells elements de joc que vulguem que estiguin sincronitzats per a tots els jugadors de la partida han de comptar amb el component *NetworkIdentity*. És el component controla la identitat única del *GameObject* a través de la xarxa. Observem que presenta dues variables:

- *Server Only*: només executarà la seva funcionalitat en la instància de joc que sigui el servidor.
- *Local Player Authority*: dóna autoritat exclusiva al client que posseeix la instància original.

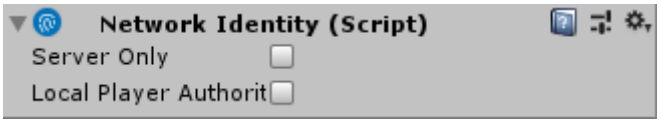

Figura 58 - Component NetworkIdentity

Els elements que presentaran aquest component seran el ***GameManager*** i el ***PlayerController***. D'aquesta manera podem sincronitzar l'estat del món i dels jugadors durant el transcurs de la partida.

Per a comprendre bé l'estructura d'aquest sistema, cal definir una sèrie de conceptes i paràmetres:

- ***SyncVar*:** variable sincronitzada en totes les instàncies de joc. Quan es modifica des de servidor, els clients rebran el paràmetre modificat.
- ***Hook*:** funció auxiliar que es crida quan una *SyncVar* ha estat modificada.
- ***Command*:** funció cridada des del client i executada en el servidor.
- ***ClientRPC*:** funció cridada des de servidor i executada en tots els clients.

Per a poder sincronitzar correctament totes les instàncies de joc, cal que els canvis en les *SyncVars* i les crides a *ClientRPC* (*Remote Procedure Calls*) s'efectuïn des del servidor (en aquest cas, el *Host*). Per a fer-ho, comprovem si el client que efectua la crida és també el servidor. Si és així, els canvis es realitzen directament.

```
public void SpawnReady()
{
    Vector2 spawnPos = uiController.SpawnReadyUI();

    // Check if we are the Server
    if (!isServer)
    {
        // If not, call Command
        localPlayer.CmdPlayerReady(spawnPos);
    }
    else
    {
        // We are the Server execute code directly
        playersReady++;
        localPlayer.currentMapPos = spawnPos;

        if (playersReady == players.Count)
            ServerEnterGameState(GameState.MATCH_START, "Match Starting");
    }
}
```

Figura 59 – Exemple de comprovació de servidor

En el cas que sigui un client el que vulgui realitzar canvis, hem de crear un *Command* que contingui la funcionalitat desitjada. Aquest *Command* s'executarà en el servidor i així sincronitzarem totes les instàncies de joc.

```
[Command]
public void CmdPlayerReady(Vector2 spawnPos)
{
    GameManager.Instance.playersReady++;
    currentMapPos = spawnPos;

    if (GameManager.Instance.playersReady == GameManager.Instance.players.Count)
        GameManager.Instance.ServerEnterGameState(GameManager.GameState.MATCH_START, "match starting");
}
```

Figura 60 - Exemple de Command

En el diagrama següent podem comprendre millor aquest procés.

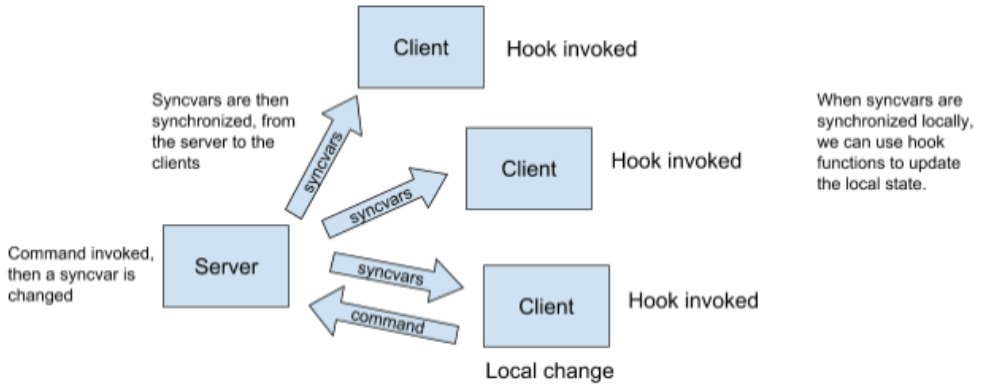

Figura 61 - Diagrama de flux de sincronització en línia

Així doncs passarem a detallar el funcionament del ***GameManager*** en relació a la connectivitat en línia:

- Presenta dues *SyncVars* principals:
  1) ***gameState*:** per controlar en cada client l'estat de la partida.
  2) ***activePlayerIndex*:** índex de la llista de jugadors que indica quin és el jugador actiu.
- Quan ha de canviar d'estat, es crida a una *ClientRPC* que, en funció del nou estat, executarà les accions corresponents. A partir d'aquesta crida ens assegurem que l'estat de la partida en tots els clients estigui sincronitzat.

- Quan un jugador finalitza el seu torn, aquest incrementa el valor de l'*activePlayerIndex* i es notifica al servidor d'un final de torn. per a que cridi a la corresponent *ClientRPC* per finalitzar el torn del jugador actual i donar pas al següent jugador.

La funcionalitat en línia del ***PlayerController*** segueix els mateixos esquemes:

- Presenta 3 *SyncVars* principals:
  1) ***inTurn*****:** per saber si el jugador es troba en el seu torn actiu o no.
  2) ***state*****:** per a sincronitzar les accions de cada jugador (*Attack*, *Movement*, *Idle*, *Dead*, etc.)
  3) ***currentMapPos*****:** punt 2D que indica la casella actual del mapa en la que es troba.
- *NetworkTransform*: component de *Unity* per a sincronitzar de manera automàtica la posició i rotació del personatge.

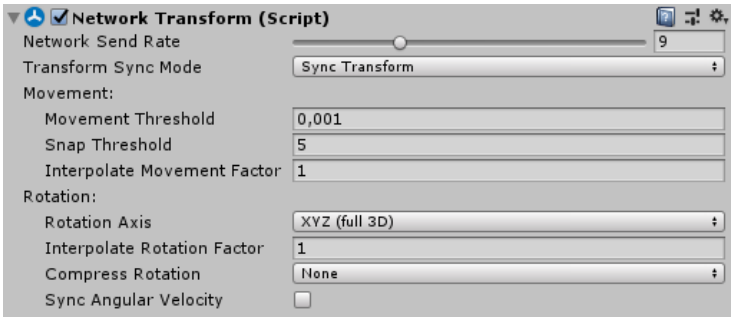

Figura 62 - Component NetworkTransform

- Quan un jugador realitza una acció que ha de ser percebuda per la resta de clients (llançament d'una habilitat, pas de torn, canvi de casella actual...), es notifica al servidor (a través d'un *Command*) si és necessari i aquest realitzarà els canvis pertinents per actualitzar l'estat i variables del jugador en tots els clients.
- **Distinció dels canvis locals:** no tots els canvis i accions que un jugador efectua s'han de transmetre a la resta de clients. Un exemple seria la modificació de la barra de vida de la UI d'un jugador quan rep mal.

  Per a poder aplicar aquests canvis locals faig ús de la variable "*isLocalPlayer*" que ens ofereix el *NetworkManager*.

```
public void GetDamage(float dmg)
{
    // Reduce Players Health
    healthPoints = healthPoints - dmg;

    // If it's local player, Update the UI hp bar
    if (isLocalPlayer)
        GameManager.UIController.UpdateHealth(healthPoints);

    // Set dead state if HP less or equal to 0
    if (healthPoints <= 0.0f)
    {
        SetState(State.DEAD);
    }

}
```

Figura 63 - Exemple d'aplicació de canvis locals

Flux de partida:

1) Els jugadors comencen indicant la posició del mapa en la que volen començar.

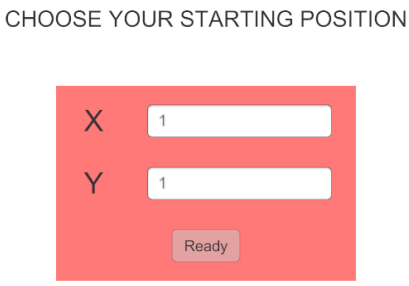

Figura 64 - Menu de selecció de casella inicial (provisional)

2) La partida comença quan tots han seleccionat una posició. El primer jugador de la llista de *players* juga el seu torn, i quan acaba cedeix el torn al següent jugador de la llista, i així successivament. L'últim jugador de la llista li tornarà a donar el torn al primer.

3) Quan un jugador mor, se li mostra la pantalla de mort i té dues opcions: tornar al menú principal o sortir del joc.

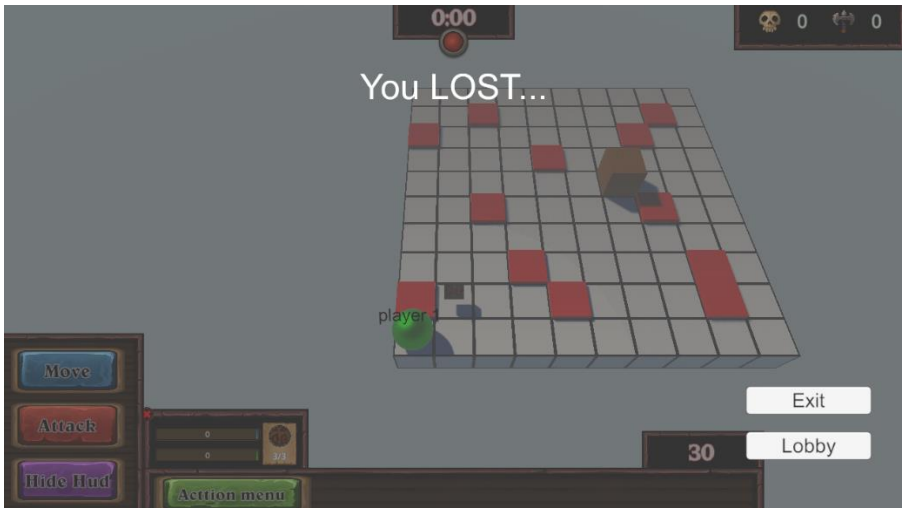

Figura 65 - Pantalla de mort (provisional)

4) La partida acaba en el moment en que només queda un jugador viu, llavors se li mostra la pantalla de victòria i finalitza la partida.

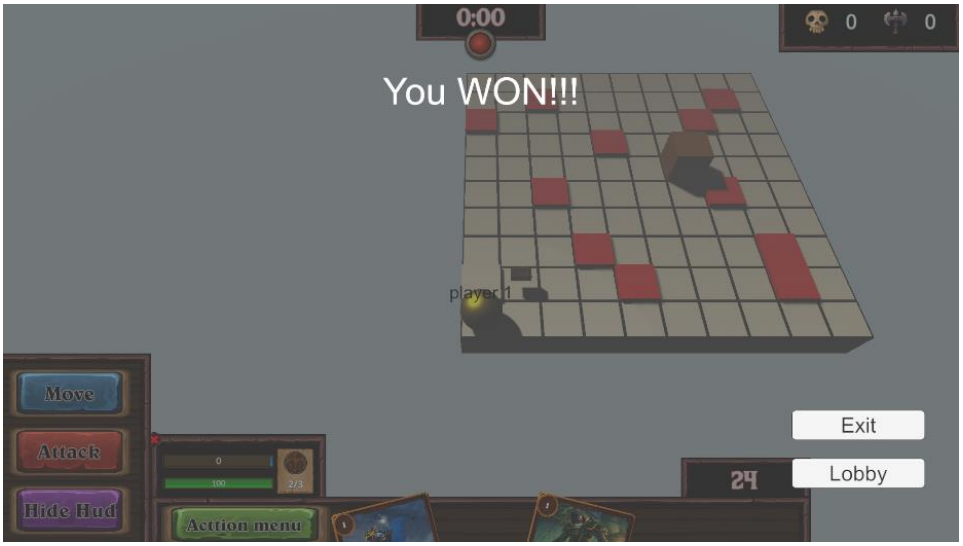

Figura 66 - Pantalla de victòria (provisional)

5.4. Estat actual del projecte i desviacions en la planificació (1 de maig de 2019)

Durant aquest període de desenvolupament m'he trobat amb dos grans reptes:

1) **Sistema de cobertures:** arribar a una solució òptima i consistent em va portar més temps de recerca i implementació de l'esperat
2) **Connectivitat Online:** sens dubte aquest és el major repte amb el que m'he trobat. Traslladar la funcionalitat d'un joc *Single Player* a un Multijugador en línia m'ha fet reorganitzar tota l'estructura base. A més, a mesura que implementava la funcionalitat de *Matchmaking* i Sincronització del joc anaven sorgint problemes amb la informació que rebia el servidor i la que enviava als clients. Solucionar tots aquests problemes i acabar d'implementar un sistema de *Networking* consistent ha requerit més temps de l'esperat.

Un cop superats aquests reptes m'he centrat en acabar d'implementar la funcionalitat principal del joc. D'aquesta manera s'ha allargat el període de temps de l'*Alpha* i endarrerit les següents fases. Aquest endarreriment no suposa un problema crític per a la finalització del projecte ja que havíem contemplat període de marge entre la finalització del projecte i l'entrega final.

En aquest link podem observar els canvis en el Diagrama de Gantt.

5.5. Programació d'habilitats

Actualment s'han implementat **dos tipus** de cartes: cartes bàsiques i cartes de classe.

- **Cartes bàsiques:** qualsevol classe pot fer-ne ús.
  - *Heal*: cura X punts de vida.
  - *Shield*: dóna X punts d'escut.
- **Cartes de classe:** només poden ser usades per la classe específica. Aquestes poden tenir diferents utilitats:
  - *Cartes d'atac:* fan mal a tot jugador situat en l'àrea d'efecte.
  - *Cartes de moviment:* permeten al jugador posicionar-se en una nova casella del mapa.
  - *Cartes de potenciació:* atorguen algun tipus de benefici al jugador que usa aquesta habilitat (invulnerabilitat, etc.)

```
// Abilities
#region Abilities

// Iterates all the area of effect tiles ...

// ...to apply damage if a player is above
public void ApplyDamage()...

// ...to recover HP if a player is above
public void RecoverHP()...

// ...to recover gain Shield if a player is above
public void GainShield()...

// Sets the player to the selected tile
public void Teleport()...

// Apply an invulnerability buff
public void Thornmail()...
```

Figura 67 - Llistat dels possibles efectes de les habilitats

Cada habilitat consta de dos elements principals: un sistema de partícules (creat pel meu company Sergio) i un efecte d'àudio. L'***AbilityController*** és el sistema que controla quines partícules i efecte d'àudio (apartat visual) es generen al llençar les cartes.

AbilityController:

Consta de 3 elements principals:

- *abilityVFX:* diccionari de *prefabs* amb les partícules de cada una de les habilitats.
- *abilitySounds:* diccionari d'àudios de cada una de les habilitats.
- *reachedDest:* booleà usat per determinar quan una habilitat ha acabat la seva funcionalitat (p.e: si es un projectil, *reachedDest* serà *true* quan hagi arribat a la casella destí).

Aquests dos diccionaris funcionen com a llistes amb la peculiaritat que cada element consta d'una *key* (identificador únic), que és el nom de la carta usada. D'aquesta manera podem saber quin sistema de partícules i efecte d'àudio crear en cada cas.

```
public void CastAbility(string abilityName, Vector3 initpoint, Vector3 endPoint, NetworkIdentity caster)
{
    GameObject abilityGO = Instantiate(abilityVFX[abilityName], initpoint, Quaternion.identity);

    // Visual Effect
    Ability ability = abilityGO.GetComponent<Ability>();
    ability.SetUp(abilityName, endPoint, caster.GetComponent<PlayerController>());

    // Audio Effect
    caster.GetComponent<AudioSource>().clip = abilitySounds[abilityName];
    caster.GetComponent<AudioSource>().Play();
}
```

Figura 68 - Generació d'habilitats (partícules i àudio)

Cada *prefab* d'habilitat incorpora un script *Ability* que controla el seu comportament.

Ability:

El *type* d'habilitat determinarà el seu funcionament principal:

- **FROM_PLAYER:** projectil que surt des del jugador que usa l'habilitat fins al seu destí, la casella seleccionada.
- **FROM_ABOVE:** projectil que cau perpendicularment sobre la casella seleccionada des d'una altura X.
- **MELEE:** atacs en àrea generats en la casella del jugador que usa la carta.
- **BUFF:** habilitats persistents al llarg de X torns.

Els altres paràmetres permeten la funcionalitat correcta de les habilitats són:

- *origin*: posició en el món on es crearà el sistema de partícules.
- *destination*: posició final a la que haurà d'arribar l'habilitat (només per a tipus *FROM_PLAYER* i *FROM_ABOVE*).
- *currDistance* i *travelDistance*: mesures de distància actual recorreguda i distància total per a saber quan el projectil ha arribat al seu destí.

```
public void SetUp(string ability, Vector3 finalPos, PlayerController caster)
{
    // Get a player reference
    player = caster;

    switch (type)
    {
        case Type.NONE:
            break;
        case Type.FROM_PLAYER:
            {
                height = 2.0f;

                origin = new Vector3(transform.position.x, transform.position.y + height, transform.position.z);
                destination = finalPos;
                destination.y = destination.y + height;

                transform.position = origin;
                transform.LookAt(destination);

                currDistance = 0.0f;
                travelDistance = Vector3.Distance(transform.position, destination);
                break;
            }
        case Type.FROM_ABOVE:
            ...
        case Type.MELEE:
            ...
        case Type.BUFF:
            ...
        case Type.BASIC:
            ...
        default:
            break;
    }
    setup = true;
}
```

Figura 69 - Configuració d'habilitat en funció del seu tipus

Així doncs observem que cada carta consta d'un **apartat visual** (Habilitat) amb les seves partícules i efectes, i un **apartat lògic** (Efecte) que aplicarà una modificació en les estadístiques (vida, escut, etc.) en els jugadors als que afecti un cop l'habilitat hagi arribat al seu destí.

Figura 70 – Diagrama de Flux d'una Habilitat

5.6. Programació d'elements de l'entorn

Els sistemes principals mencionats anteriorment formen el nucli del joc, ara bé, per complementar i donar riquesa a aquest prototip i apropar-lo més a l'essència d'un *Battle Royale* estratègic s'han desenvolupat dos petits sistemes que augmentaran el grau d'interacció Jugador-Entorn i aportaran un cert factor de progressió a la partida.

5.6.1. *Spawn* d'Objectes en el mapa

Podem trobar-nos amb dos elements diferents repartits pel mapa:

- **Cobertures:** els obstacles que impedeixen el moviment i llançament d'habilitats dels jugadors.
- **Cofres:** els jugadors accediran a ells per obtenir noves cartes. Per a obrir un cofre s'hauran de moure fins a una casella adjacent a aquest (nord, sud, est i oest). Un cop situats, se'ls mostrarà per pantalla una finestra amb tres cartes i podran escollir si incorporar-les a la seva baralla o no.

Figura 71 - Elements d'entorn (cofres i cobertures)

El sistema de generació d'aquestes cartes funciona per mitjà de **percentatges**, d'aquesta manera, es deixa a l'atzar (controlat) l'aparició de cartes bàsiques i/o de classe. En funció de la classe, les cartes que podran aparèixer seran diferents. Si en un futur es decideix

implementar un sistema de raresa de cartes, cartes úniques, noves habilitats... serà fàcil d'integrar amb aquest sistema.

Figura 72 – Selector de noves cartes

Aquesta finestra de selecció de cartes també apareixerà quan un jugador acabi amb un contrincant.

El *MapController* conté una llista de *tiles* per a saber quines seran les caselles on apareixeran els cofres i una altra llista per a les cobertures. Al inici de partida, doncs, es carregaran aquests tots els cofres i cobertures en cada *tile* corresponent.

5.6.2. *Death Zone*

Un dels elements característics dels *Battle Royale*, la zona que delimita l'àrea jugable, que a mesura que transcorre la partida es va reduint. Fora d'aquesta àrea, els jugadors van rebent mal progressivament fins que moren.

A l'hora de plasmar aquesta idea en el nostre prototip podem desglossar el seu funcionament en 4 característiques principals:

1) ***Initial Damage:*** dany base a l'inici de la partida.

2) ***Incremental Damage:*** els punts de mal que incrementa cada cop que la *death zone* s'expandeix.
3) ***Propagation Rate:*** cada quantes rondes completes (una ronda equival a que tots els jugadors de la partida hagin jugat el seu torn) la zona s'expandirà.
4) ***Min Distance:*** delimita les caselles mínimes que ha de deixar de marge com a zona jugable. A partir d'aquest punt no es pot expandir més.

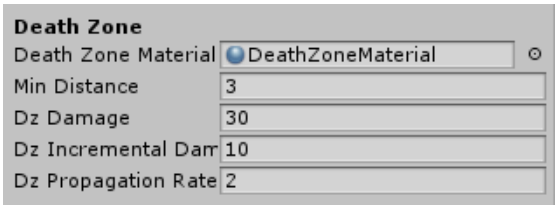

Figura 73 - Variables editables de la Death Zone

Figura 74 - Death Zone (màxima expansió)

El funcionament d'aquesta Death Zone és simple: cada X rondes es va expandint fins arribar al seu límit establert. Quan un jugador inicia el seu torn sobre una tile de la death zone, aquest rebrà mal, amb la possibilitat de morir si la seva vida queda reduïda a 0.

5.7. Evolució de la UI

La pantalla de selecció de posició inicial s'ha actualitzat. Apareixen quatre botons clicables que representen cadascuna de les **zones** en les que es divideix el mapa. Així doncs, quan tots els jugadors hagin seleccionat la seva zona, el joc els situarà en una casella disponible de manera aleatòria dins la zona que han escollit. El *MapController* contempla cadascuna de les àrees i caselles disponibles per a la correcta col·locació del jugador en el mapa (per evitar solapaments entre jugadors, cobertures o cofres).

Figura 75 - Pantalla de selecció de zona

Les pantalles de mort i final de partida també s'han actualitzat a una segona versió.

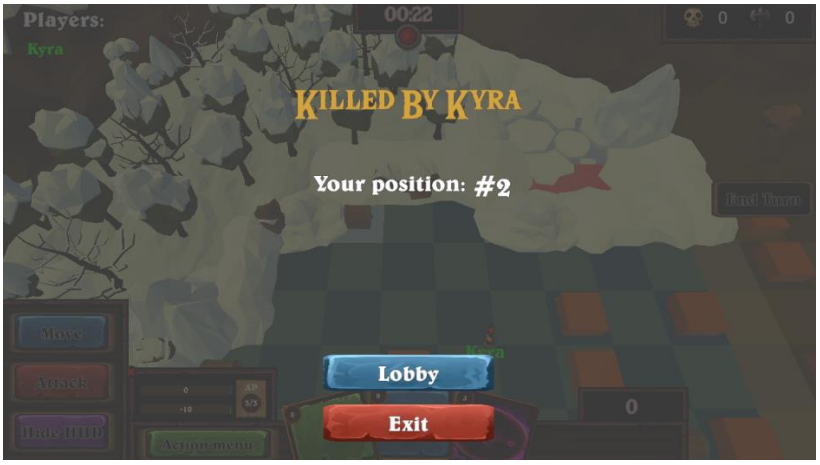

Figura 76 - Pantalla de derrota

Figura 77 - Pantalla de victòria

5.8. Fase de Testeig i *Bugfixing*

Un cop tota la funcionalitat del joc ha sigut implementada, s'ha procedit a testejar el joc tant de manera interna com externa.

Bugfixing

Es van realitzar sessions internes de testeig (entre els creadors del prototip, en Sergio i jo) on l'objectiu era trobar el màxim d'errors possibles. La prioritat es donava a aquells *bugs* que bloquejaven el *gameplay*, impedint el progrés de la partida. La majoria de *bugs* d'aquest tipus tenien relació amb el sistema de *Networking* (un jugador moria i no es passava al següent torn, impedint el progrés de la partida). També s'han arreglat errors amb els elements de l'entorn (cobertures i cofres) i altres *bugs* menors (errors en textos, imatges...).

Fase de Testeig

Es van organitzar diverses sessions de *Playtesting* grupals en que quatre usuaris jugaven una partida. Al finalitzar, omplien el formulari amb la seva opinió.

Aquest formulari ha estat dissenyat pel meu company Sergio i compta amb una sèrie de preguntes relacionades amb diferents aspectes del joc: *User Experience*, Jugabilitat, Ambientació, Sensacions i Conclusions.

Amb les dades recollides s'han modificat diferents aspectes del joc:

- Estadístiques de les cartes (Cost, Dany, Rang)
- Estadístiques dels personatges (Rang de moviment, AP)
- Característiques de partida (temps de torn, propagació de la *Death Zone*...)

Aquests *playtestings* ens han servit per a afavorir l'experiència de joc i jugabilitat, implementant millores visuals (més feedback i indicacions per al jugador), reorganitzacions de la UI, etc.

En aquest link podem veure les respostes dels usuaris.

6. CONCLUSIONS I TREBALLS FUTURS

6.1 Anàlisi d'Objectius

Primer de tot passem a analitzar els objectius que es van proposar inicialment i el seu estat un cop ha finalitzat el projecte:

- **Controlador de personatge:** s'ha desenvolupat un mòdul que permet al jugador controlar les funcions principals del seu personatge (moure's, utilitzar cartes, etc.) amb una distinció per classes (*Warrior, Mage, Rogue*).
- **Mapa:** implementat un sistema de control de caselles i *pathfinding* per a generar àrees d'habilitat i moviment, i generació d'elements d'entorn (cofres, cobertures i *Death Zone*).
- **Interfície d'Usuari:** implementada la interfície general del joc que permet al jugador interactuar i veure tota la informació necessària del joc (HUD). S'han integrat les diverses pantalles per les quals el jugador accedeix a la partida i finalitza la seva sessió de joc (Menú principal, *Network Lobby*, Selecció de *spawn* inicial, Pantalla de victòria/derrota).
- **Connectivitat a Internet:** integrat un sistema de *Networking* basat en el protocol Client-Host ofert per la HLAPI de UNET. Els elements que disposen de funcionalitat en línia principalment són: el *GameManager* i el *PlayerController*.

- **Sistema de Cartes "Col·leccionables":** implementada la creació de cartes a partir de *Scriptable Objects* i un sistema de gestió de baralla (*Deck*) que controla les cartes del jugador (com adquirir cartes, robar, utilitzar-les, mesclar...). Cada carta compta amb la seva habilitat visual (partícules i efectes de so) i la seva part lògica (efecte, p.e: fer 10 punts de mal, guanyar 10 HP...).
- ***Game Loop*:** el *GameManager* conté tota la funcionalitat de partida necessària per a la progressió d'aquesta (control dels torns, control de l'estat del joc...)

6.2 Anàlisi de Planificació

En un projecte de mitja-llarga durada com aquest no és fàcil estimar amb exactitud el temps i esforç necessaris per a desenvolupar cada tasca. Per això, marcar un pla de contingències és vital per a superar els obstacles sorgits al llarg del desenvolupament.

Durant aquest projecte hi ha hagut una sèrie de desviacions i canvis degut als dos grans obstacles mencionats en l'apartat 5.4 (Sistema de *Networking* i Sistema de Cobertures). Així doncs hem aplicat una sèrie de contingències que han donat els seus resultats:

- **Personatges estàtics (sense animacions):** el temps estalviat en crear i programar el sistema d'animacions s'ha dedicat en completar el sistema de *Networking.*
- **Reaprofitament de funcionalitat de les cartes:** l'estructura separada entre la lògica i l'apartat visual facilita la creació de noves cartes, ja que la programació ja està implementada (apartat 5.5). Només cal crear un nou sistema de partícules que representi visualment l'habilitat.
- **Simplificació d'Interfície d'Usuari:** s'han deixat de banda les animacions de la UI per tal de tenir temps en acabar el sistema i deixar-lo el més consistent possible.

Des d'un inici tenia clar centrar-me en la base de tots els sistemes principals per deixar un prototip simple però consistent. La funció d'un prototip és la de provar i testejar totes les mecàniques del joc per així passar al desenvolupament final.

6.3 Ampliacions i millores

A partir de l'estat actual del projecte es podrien implementar una sèrie de millores que aproparien aquest prototip a l'estat d'un joc acabat:

- **Ampliar la varietat de cartes** amb noves habilitats per donar als jugadors diferents estratègies a l'hora de construir la seva baralla i enfrontar-se als seus rivals.
- Ampliar el sistema de cartes amb un **sistema de raresa** per enriquir la jugabilitat i donar als jugadors la possibilitat d'adquirir cartes més poderoses, augmentant les seves possibilitats de guanyar.
- Integrar **proceduralitat** (aleatorietat) en la disposició de cobertures i cofres en el mapa. D'aquesta manera afavorim la rejugabilitat en cada partida.
- Ampliar el mapa amb **caselles especials** (dificultant el moviment, otorgant stats especials als jugadors...) per enriquir la jugabilitat i tàctiques dels jugadors.
- Implementar un **sistema de progressió** fora de partida (nivell de jugador, missions diàries, reptes setmanals...) per incentivar els usuaris a seguir jugant i allargar la vida del joc.
- **Millorar l'apartat visual** (amb animacions, efectes d'àudio etc.) per augmentar la qualitat del projecte. Els jocs que surten al mercat han d'entrar bé pels ulls i tenir un mínim de qualitat visual.
- Millorar el **Sistema *Matchmaking*** amb servidors dedicats que aguantin un gran nombre de jugadors simultanis i que controli adequadament les desconnexions dels jugadors durant una partida.

6.4 Observacions finals

Al llarg d'aquest projecte he après lliçons valuoses en relació als diversos apartats en què he hem dividit el projecte:

Sistema Networking

- El sistema que ofereix Unity (*UNet Services*) està en desús i acabarà desapareixent a partir de les versions del 2022. Ha **quedat obsolet** i no cobreix

totes les funcionalitats i necessitats que els desenvolupadors necessiten actualment (optimització, seguretat, millora d'infraestructura...).

Cal dir que aquest sistema és **fàcil i ràpid** d'implementar i cobreix correctament les necessitats del nostre prototip. Ara bé, si es vol crear un joc per a llençar-lo al mercat optaria per una alternativa com Photon PUN i Photon Bolt, que presenten unes infraestructures més estables i segures.

- A l'hora de desenvolupar un videojoc online, des del primer moment s'ha d'estructurar el codi **encarat al sistema de *Networking*** implementat. Així ens estalviem temps i esforç en reestructurar el codi si inicialment hem implementat les mecàniques encarades a un joc *Single Player*.
- **Separar** la funcionalitat **online** de la funcionalitat **local** dels elements que hagin de sincronitzar-se a través de la xarxa. Així podem accedir més fàcilment a cada apartat i organitzar-ho millor.

Apartat visual

- El ***feedback* visual** (partícules, notificacions, àudios...) de les accions del jugador és vital per a que entengui què està passant en qualsevol moment de la partida i no se senti frustrat.

Sistema de Cartes

- Els ***Scriptable Objects*** faciliten molt la creació de noves cartes simplement modificant els paràmetres establerts.

Controlador del Mapa

- Tractar la lògica d'un joc estratègic d'aquest tipus en **dues dimensions** és un encert (així es té un bon control sobre cada una de les caselles del mapa).

Per finalitzar, m'agradaria dir que la realització d'aquest projecte no hagués sigut possible sense la bona **comunicació i organització** entre el meu company Sergio Sáez i jo. Hem sabut sincronitzar les nostres tasques i superar els riscos i obstacles que han anat apareixent al llarg del desenvolupament.

7. BIBLIOGRAFIA

- **Vandal**. *¿Quién invento los juegos Battle Royale?¿Cuás fue el primero?* - Consultat 25 Feb, 2019. *https://vandal.elespanol.com/noticia/1350708240/quien-invento-los-juegos-battle-royale-cual-fue-el-primero/*
- **Wikipedia**. *Juego de cartes col·leccionables* – Consultat 23 Feb, 2019. *https://es.wikipedia.org/wiki/Juego_de_cartas_coleccionables*
- **Wikipedia**. *Digital collectible card game* – Consultat 23 Feb, 2019. *https://en.wikipedia.org/wiki/Digital_collectible_card_game*
- **Game Designing**. *The Top 10 Video Game Engines* – Consultat 24 Feb, 2019. *https://www.gamedesigning.org/career/video-game-engines/*
- **Baboon Lab**. U*nreal Engine 4, el motor gráfico que ofrece realismo al máximo* – Consultat 24 Feb, 2019. *http://www.baboonlab.com/blog/noticias-de-marketing-inmobiliario-y-tecnologia-1/post/unreal-engine-4-el-motor-grafico-que-ofrece-realismo-al-maximo-23*
- **Gaffer on Games**. *What Every Programmer Needs To Know About Game Networking* – Consultat 25 Feb, 2019. *https://gafferongames.com/post/what_every_programmer_needs_to_know_about_game_networking/*
- **Luis Marketing Online.** *Modelos de ingresos para rentabilizar una App* – Consultat 9 Mar, 2019. *https://luismarketingonline.es/modelos-de-ingresos-para-rentabilizar-una-app/*
- **Unity - Documentation.** *Multiplayer and Networking* – Consultat 1 Mar, 2019. *https://docs.unity3d.com/Manual/UNet.html*
- **Unity - Forum.** *Network Lobby creation* – Consultat 2 Mar, 2019. *https://answers.unity.com/questions/1444571/how-to-create-a-custom-network-lobby-manager-with.html*

- **A Medium Corporation.** *Turn-based Multiplayer Games in Unity3D* – Consultat 5 Mar, 2019. *https://medium.com/@div5yesh/turn-based-multiplayer-games-in-unity3d-using-unet-abcd8360ddd5*
- **Fabulous Adventures in Coding.** *Shadowcasting in C#* – Consultat 20 Mar, 2019. *https://blogs.msdn.microsoft.com/ericlippert/2011/12/12/shadowcasting-in-c-part-one/*
- **Wordpress, DirkKok.** *FOV using recursive shadowcasting in c#* – Consultat 21 Mar, 2019. *https://dirkkok.wordpress.com/2008/02/17/fov-using-recursive-shadow-casting-in-c/*
- **Youtube.** *Make a quick Card System* – Consultat 5 Abr, 2019. *https://www.youtube.com/watch?v=UEkCCn3nQpo*
- **Unity - Documentation.** *Spawning GameObjects* – Consultat 5 Maig, 2019. *https://docs.unity3d.com/Manual/UNetSpawning.html*
- **Photon.** *Welcome page* – Consultat 5 Juny, 2019. *https://www.photonengine.com/en/pun*

ANNEXOS

A. Anàlisi Financer

A.1 Anàlisi d'Ingressos

MILLOR ESCENARI						
Suposicions				*Game Launch*		
ARPPU	10,00 €			**GEN**	**FEB**	**MAR**	**ABR**
Rati de conversió	10%		**Nous Usuaris**	350	900	500	320
			Usuaris Totals	**350**	**1250**	**1750**	**2070**
			Ingressos	350,00 €	900,00 €	500,00 €	320,00 €
			Ingressos Totals	**350,00 €**	**1.250,00 €**	**1.750,00 €**	**2.070,00 €**

ESCENARI REGULAR						
Suposicions				*Game Launch*		
ARPPU	7,50 €			**GEN**	**FEB**	**MAR**	**ABR**
Rati de conversió	10%		**Nous Usuaris**	300	570	330	250
			Usuaris Totals	300	870	1200	1450
			Ingressos	225,00 €	427,50 €	247,50 €	187,50 €
			Ingressos Totals	**225,00 €**	**652,50 €**	**900,00 €**	**1.087,50 €**

PITJOR ESCENARI						
Suposicions				*Game Launch*		
ARPPU	5,00 €			**GEN**	**FEB**	**MAR**	**ABR**
Rati de conversió	5%		**Nous Usuaris**	250	300	150	120
			Usuaris Totals	**250**	**550**	**700**	**820**
			Ingressos	62,50 €	75,00 €	37,50 €	30,00 €
			Ingressos Totals	**62,50 €**	**137,50 €**	**175,00 €**	**205,00 €**

A.2 Anàlisi de Costos

													Game Launch			
	GEN	FEB	MAR	ABR	MAI	JUN	JUL	AGO	SEP	OCT	NOV	DES	GEN	FEB	MAR	ABR
COSTOS																
Programmer - Jordi	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €
Game Designer - Sergio	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €	1.200,00 €
Humble Bundle: Fantasy Art RPG Pack	18,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Humble Bundle: Fantasy UI RPG Pack	0,00 €	18,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Polygon: Adventure pack	0,00 €	14,29 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Polygon: Fantasy Characters	0,00 €	9,28 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €
Depreciació	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €	80,54 €
Costos Mesnuals	2.498,54 €	2.522,11 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €	2.480,54 €
Costos Acumulatius	**2.498,54 €**	**5.020,65 €**	**7.501,20 €**	**9.981,74 €**	**12.462,28 €**	**14.942,82 €**	**17.423,36 €**	**19.903,90 €**	**22.384,45 €**	**24.864,99 €**	**27.345,53 €**	**29.826,07 €**	**32.306,61 €**	**34.787,15 €**	**37.267,70 €**	**39.748,24 €**

A.3 Anàlisi del Balanç

													Game Launch			
	GEN	FEB	MAR	ABR	MAI	JUN	JUL	AGO	SEP	OCT	NOV	DES	GEN	FEB	MAR	ABR
INGRESSOS																
Ingressos mensuals	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	0,00 €	225,00 €	427,50 €	247,50 €	187,50 €
Ingressos acumulatius	**0,00 €**	**0,00 €**	**0,00 €**	**0,00 €**	**0,00 €**	**0,00 €**	**0,00 €**	**0,00 €**	**0,00 €**	**0,00 €**	**0,00 €**	**0,00 €**	**225,00 €**	**652,50 €**	**900,00 €**	**1.087,50 €**
COSTOS																
Costos mensuals	-2.498,54 €	-2.522,11 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €
Costos acumulatius	**-2.498,54 €**	**-5.020,65 €**	**-7.501,20 €**	**-9.981,74 €**	**-12.462,28 €**	**-14.942,82 €**	**-17.423,36 €**	**-19.903,90 €**	**-22.384,45 €**	**-24.864,99 €**	**-27.345,53 €**	**-29.826,07 €**	**-32.306,61 €**	**-34.787,15 €**	**-37.267,70 €**	**-39.748,24 €**
BALANÇ																
Balanç mensual	-2.498,54 €	-2.522,11 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.480,54 €	-2.255,54 €	-2.053,04 €	-2.233,04 €	-2.293,04 €
Balanç cumulatius	**-2.498,54 €**	**-5.020,65 €**	**-7.501,20 €**	**-9.981,74 €**	**-12.462,28 €**	**-14.942,82 €**	**-17.423,36 €**	**-19.903,90 €**	**-22.384,45 €**	**-24.864,99 €**	**-27.345,53 €**	**-29.826,07 €**	**-32.081,61 €**	**-34.134,65 €**	**-36.367,70 €**	**-38.660,74 €**

A.4 Anàlisi de la Depreciació

Assets	Cost	Temps d'Amortització		Amortització anual	Depreciació mesnual
		Anys	Mesos		
Jordi					
Ordinador Portàtil	1.250,00 €	3	36	416,67 €	34,72 €
Monitor	200,00 €	4	48	50,00 €	4,17 €
Teclat	80,00 €	4	48	20,00 €	1,67 €
Mouse	45,00 €	2	24	22,50 €	1,88 €
Sergio					
PC	1.000,00 €	3	36	333,33 €	27,78 €
Monitor	220,00 €	2	24	110,00 €	9,17 €
Teclat	40,00 €	5	60	8,00 €	0,67 €
Mouse	30,00 €	5	60	6,00 €	0,50 €

Depreciació	
Ordinadors	62,50 €
Monitors	13,33 €
Teclats	2,33 €
Mouses	2,38 €
Total	**80,54 €**